目　录

第 1 章　MATLAB 6.0 概述 1

1.1　MATLAB 简介 1

1.1.1　MATLAB 的发展历程 1

1.1.2　MATLAB 6.0 对系统环境的要求 2

1.1.3　MATLAB 6.0 软件的安装 3

1.2　MATLAB 6.0 新特性 5

1.2.1　MATLAB 语言的发展 5

1.2.2　MATLAB 6.0 的开发环境 5

1.2.3　MATLAB 的数学计算能力 6

1.2.4　MATLAB 6.0 的其他新特点 7

1.3　SIMULINK 4.0 概述 8

1.3.1　什么是 SIMULINK 8

1.3.2　SIMULINK 4.0 的组成 8

1.3.3　SIMULINK 4.0 的特点 9

第 2 章　熟悉 MATLAB 6.0 环境 10

2.1　MATLAB 6.0 的桌面环境 10

2.2　MATLAB 基本指令和用法 11

2.2.1　数值、变量和表达式 11

2.2.2　向量运算 12

2.2.3　矩阵的简单运算 13

2.2.4　矩阵的特殊运算 15

2.2.5　元胞数组 16

2.2.6　结构数组 18

2.2.7　数据的图形显示 20

2.2.8　命令窗口基本指令 21

2.3　工作空间 22

2.3.1　工作空间简介 22

2.3.2　基本指令 22

2.3.3　使用工作空间浏览器 23

2.3.4　工作空间的保存 23

2.4　路径设置 24

2.4.1　路径设置简介 24

2.4.2　目录的设置 24

2.4.3　路径浏览器的使用 25

2.5 M 文件的编写与调试......25
2.5.1 M 文件编辑器......25
2.5.2 MATLAB 控制流结构......26
2.5.3 M 函数文件......28
2.5.4 变量的作用范围......28
2.5.5 M 文件的调试......28
2.6 在线演示和帮助......30
2.6.1 在线引导......30
2.6.2 演示程序......30
2.6.3 帮助系统......31
第 3 章 SIMULINK 4.0 概述......34
3.1 SIMULINK 4.0 导引......34
3.1.1 SIMULINK 4.0 的安装......34
3.1.2 SIMULINK 4.0 的启动......34
3.1.3 SIMULINK 4.0 的工作环境......35
3.1.4 SIMULINK 4.0 的演示程序......35
3.2 SIMULINK 4.0 的组成......36
3.2.1 应用工具箱......36
3.2.2 实时工作室......36
3.2.3 状态流模块......36
3.2.4 扩展的模块集......37
3.2.5 SB2SL 工具......37
3.3 SIMULINK 中的基本概念......37
3.3.1 模块与模块框图......37
3.3.2 信号......38
3.3.3 求解器......39
3.3.4 子系统......39
3.3.5 零点穿越......39
3.4 SIMULINK 的常用工具......40
3.4.1 仿真加速器......40
3.4.2 模型比较工具......42
3.4.3 仿真统计表......43
3.5 SIMULINK 环境的设置......43
3.5.1 MATLAB 环境设置对话框......43
3.5.2 SIMULINK 环境的设置......43
3.6 一个简单的例子......44
3.6.1 开始......44
3.6.2 创建模型......45
3.6.3 仿真配置......46
3.6.4 启动仿真......46

3.6.5 结果 47
第 4 章 模型的创建 48
4.1 模型和模型文件 48
4.1.1 SIMULINK 模型的概念 48
4.1.2 模型文件的创建和修改 48
4.1.3 模型的打印 49
4.1.4 模型的注释 50
4.2 模块操作 50
4.2.1 模块的基本概念 50
4.2.2 模块的基本操作 51
4.2.3 模块的向量化与标量扩展 54
4.2.4 模块的参数设置 55
4.3 模型中的信号 56
4.3.1 概述 56
4.3.2 确定输出信号的维数 57
4.3.3 信号属性的设置 57
4.4 信号线操作 58
4.4.1 绘制信号线 58
4.4.2 信号线的移动与删除 58
4.4.3 信号线的分支 59
4.4.4 信号线的显示属性 59
4.4.5 注释信号线 60
4.5 数据类型与数据对象 60
4.5.1 SIMULINK 中的数据类型 60
4.5.2 数据对象概述 61
4.5.3 创建数据对象 62
4.5.4 在 SIMULINK 模型中使用数据对象 63
4.5.5 数据类的创建 63
4.5.6 SIMULINK 数据浏览器 65
4.6 模型创建指令介绍 66
4.6.1 导引 66
4.6.2 指令详解 67
4.7 模块库与连接 75
4.7.1 导引 75
4.7.2 库的创建与修改 76
4.7.3 创建对库的连接 76
4.7.4 修改具有连接的子系统 77
4.7.5 连接模块的更新与显示 77
4.7.6 浏览模块库 78
4.8 模型的查找与浏览 78

4.8 模型的查找与浏览 78
4.8.1 模型对象的查找 78
4.8.2 模型浏览器的使用 80
4.9 建模的方法与技巧 80
4.9.1 创建子系统 80
4.9.2 使用回调例程 81
4.9.3 建模时的考虑 83
4.9.4 方程的建模 83
4.9.5 快捷键介绍 85
4.10 管理模型的版本 86
4.10.1 导引 86
4.10.2 指定当前的用户 86
4.10.3 模型属性对话框 87
4.10.4 模型修改日志的创建与编辑 88
第5章 SIMULINK 仿真模块库 89
5.1 SIMULINK 库 89
5.1.1 SIMULINK 库简介 89
5.1.2 标准 SIMULINK 模块库 89
5.1.3 SIMULINK 扩展库（Simulink Extras） 90
5.2 SIMULINK 模块集 91
5.2.1 通信模块集（Communications Blockset） 91
5.2.2 面板与仪表模块集（Dials & Gauges Blockset） 94
5.2.3 数字信号处理模块（DSP Blockset） 95
5.2.4 定点模块库（Fixed-Point Blockset） 95
5.2.5 非线性控制系统设计模块集（NCD Blockset） 96
5.2.6 神经网络模块集（Neural Network Blockset） 96
5.2.7 MPC 模块集（MPC Blockset） 97
5.2.8 电力系统模块集（Power System Blockset） 97
5.3 其他辅助工具 98
5.3.1 实时窗口目标库（Real-Time Windows Target） 98
5.3.2 实时工作室（Real-Time Workshop） 98
5.3.3 状态流模块库（Stateflow） 99
第6章 SIMULINK 模块库索引 100
6.1 Source 库 100
6.1.1 Band-Limit White Noise 模块 100
6.1.2 Chirp Signal 模块 101
6.1.3 Clock 模块 102
6.1.4 Constant 模块 102
6.1.5 Digital Clock 模块 103

6.1.6 Discrete Pulse Generator 模块 103
6.1.7 From Workspace 模块 104
6.1.8 From File 模块 105
6.1.9 Pulse Generator 模块 106
6.1.10 Ramp 模块 107
6.1.11 Random Number 模块 107
6.1.12 Repeating Sequence 模块 108
6.1.13 Signal Generator 模块 109
6.1.14 Sine Wave 模块 109
6.1.15 Step 模块 110
6.1.16 Uniform Random Number 模块 111
6.2 Sinks 库 112
6.2.1 Display 模块 112
6.2.2 Scope 模块 113
6.2.3 Stop Simulation 模块 114
6.2.4 To File 模块 114
6.2.5 To Workspace 模块 115
6.2.6 XY Graph 模块 116
6.3 Discrete 库 117
6.3.1 Discrete Filter 模块 117
6.3.2 Discrete State-Space 模块 118
6.3.3 Discrete-Time Integrator 模块 119
6.3.4 Discrete Transfer Fcn 模块 122
6.3.5 Discrete Zero-Pole 模块 123
6.3.6 First-Order Hold 模块 124
6.3.7 Zero-Order Hold 模块 124
6.3.8 Unit Delay 模块 125
6.4 Continuous 库 125
6.4.1 Derivative 模块 125
6.4.2 Integrator 模块 126
6.4.3 Memory 模块 129
6.4.4 State-Space 模块 129
6.4.5 Transfer Fcn 模块 130
6.4.6 Transfer Delay 模块 131
6.4.7 Variable Transport Delay 模块 132
6.4.8 Zero-Pole 模块 133
6.5 Math 库 134
6.5.1 Abs 模块 134
6.5.2 Algebraic Constraint 模块 135
6.5.3 Bitwise Logical Operator 模块 136

6.5.4 Combinatorial Logic（组合逻辑）模块 137
6.5.5 Complex to Magnitude-Angle 模块 138
6.5.6 Complex to Real-Image 模块 139
6.5.7 Dot Product 模块 139
6.5.8 Gain 模块 140
6.5.9 Logical Operator 模块 141
6.5.10 Magnitude-Angle to Complex 模块 142
6.5.11 Math Function 模块 142
6.5.12 Matrix Gain 模块 143
6.5.13 MinMax 模块 144
6.5.14 Product 模块 145
6.5.15 Real-Image to Complex 模块 146
6.5.16 Relational Operator 模块 147
6.5.17 Rounding Function 模块 148
6.5.18 Sign 模块 148
6.5.19 Slider Gain 模块 149
6.5.20 Sum 模块 149
6.5.21 Trigonometric Function 模块 150
6.6 Nonlinear 库 151
6.6.1 Backlash 模块 151
6.6.2 Coulomb & Viscous Friction 模块 152
6.6.3 Dead Zone 模块 153
6.6.4 Manual Switch 模块 154
6.6.5 Multiport Switch 模块 154
6.6.6 Quantizer 模块 155
6.6.7 Rate Limiter 模块 156
6.6.8 Relay 模块 157
6.6.9 Saturation 模块 158
6.6.10 Switch 模块 158
6.7 Function & Table 库 159
6.7.1 Direct Look-Up Table（n-D）模块 159
6.7.2 Fcn 模块 161
6.7.3 Look-Up Table 模块 162
6.7.4 Look-Up Table（2-D）模块 163
6.7.5 Look-Up Table（n-D）模块 164
6.7.6 MATLAB Fcn 模块 165
6.7.7 Polynomial 模块 166
6.7.8 PreLook-Up Index Search 模块 167
6.7.9 Interpolation（n-D）Using PreLook-Up 模块 168
6.7.10 S-Function 模块 169

6.8 Signals & Systems 库 170
6.8.1 Bus Selector 模块 170
6.8.2 Configurable Subsystem 模块 171
6.8.3 Data Store Memory 模块 171
6.8.4 Data Store Read 模块 172
6.8.5 Data Store Write 模块 173
6.8.6 Data Type Conversion 模块 173
6.8.7 Demux 模块 174
6.8.8 Enable 模块 174
6.8.9 From 模块 175
6.8.10 Function-Call Generator 模块 176
6.8.11 Goto 模块 177
6.8.12 Goto Tag Visibility 模块 177
6.8.13 Ground 模块 178
6.8.14 Hit Crossing 模块 178
6.8.15 IC 模块 179
6.8.16 Inport 模块 179
6.8.17 Merge 模块 181
6.8.18 Model Info 模块 182
6.8.19 Mux 模块 182
6.8.20 Output 模块 183
6.8.21 Selector 模块 184
6.8.22 Subsystem 模块 184
6.8.23 Terminator 模块 184
6.8.24 Trigger 模块 184
6.8.25 Width 模块 185
6.8.26 Probe 模块 185
6.8.27 Reshape 模块 186
6.8.28 Matrix Concatenation 模块 186
第 7 章 子系统的创建与封装 188
7.1 子系统介绍 188
7.1.1 分层的建模思想 188
7.1.2 用户模块库的定制 189
7.1.3 条件子系统 189
7.1.4 一个简单的例子 189
7.2 一般子系统 192
7.2.1 什么是一般子系统 192
7.2.2 采用框选法创建一般子系统 192
7.2.3 采用 Subsystem 模块方法创建子系统 193
7.3 封装子系统 193

7.3.1 什么是封装子系统......193
7.3.2 封装子系统的创建过程......193
7.3.3 参数对话框的设置......193
7.4 条件子系统......198
7.4.1 使能子系统......198
7.4.2 触发子系统......199
7.4.3 触发-使能子系统......200
7.4.4 交替执行子系统......200
7.4.5 条件子系统小结......201
第8章 仿真模型的分析......202
8.1 模型状态的确定......202
8.1.1 导引......202
8.1.2 确定模型状态......202
8.1.3 平衡点的确定......203
8.1.4 实例......203
8.2 模型的线性化问题......204
8.2.1 线性化的数学描述......204
8.2.2 连续系统的线性化......205
8.2.3 离散系统的线性化......206
8.2.4 实例......206
8.3 代数环问题......207
8.3.1 仿真模型中的代数环......207
8.3.2 非代数环的情况......208
8.4 微分方程的求解算法......209
8.4.1 微分方程的求解......209
8.4.2 各种求解方法的比较......209
8.5 积分步长与容许误差......210
8.5.1 积分步长的选择......210
8.5.2 容许误差的设置......210
第9章 运行仿真......212
9.1 启动仿真过程......212
9.1.1 仿真入门......212
9.1.2 用菜单方式启动仿真......213
9.1.3 仿真过程的诊断......213
9.2 仿真的配置......214
9.2.1 求解器的设置......214
9.2.2 工作空间 I/O 的设置......216
9.2.3 诊断页的设置......217
9.2.4 高级属性的设置......218

9.3 优化仿真过程 220
9.3.1 介绍 220
9.3.2 提高仿真速度 220
9.3.3 提高仿真精度 221
9.4 从命令窗口中执行仿真 221
9.4.1 介绍 221
9.4.2 使用 sim 指令 222
9.4.3 仿真配置指令的使用 222
9.5 仿真结果的观察 224
9.5.1 使用示波器 224
9.5.2 使用返回变量方式 225
9.5.3 使用工作空间方式 225
第 10 章 模型的调试 227
10.1 SIMULINK 4.0 的调试环境 227
10.1.1 启动调试器 227
10.1.2 开始调试 228
10.1.3 获取在线帮助 229
10.2 调试过程 229
10.2.1 调试步骤 229
10.2.2 无条件断点的设置 230
10.2.3 无条件断点的清除 231
10.2.4 条件断点的设置 231
10.3 仿真信息的显示 232
10.3.1 模块 I/O 的显示 232
10.3.2 代数环的显示 233
10.3.3 显示系统状态 233
10.4 模型信息的显示 233
10.4.1 显示模型中模块的执行次序 233
10.4.2 根据索引号确定模块 234
10.4.3 显示模型当中的非虚拟系统 234
10.4.4 显示模型当中的非虚拟模块 234
10.4.5 显示零点穿越模块 235
10.4.6 显示代数环 235
10.4.7 显示调试器设置信息 235
10.4.8 调试命令列表 236
第 11 章 S 函数的编写 237
11.1 S 函数概述 237
11.1.1 什么是 S 函数 237
11.1.2 什么时候使用 S 函数 237

11.1.3 S函数的工作原理........238
11.1.4 S函数的基本概念........239
11.2 采用M文件编写S函数........240
11.2.1 S函数模块的创建........240
11.2.2 连续系统的S函数........243
11.2.3 离散系统的S函数........244
11.2.4 混合系统的S函数........246
11.3 采用C MEX文件编写S函数........248
11.3.1 概述........248
11.3.2 简单C MEX文件的创建........248
11.3.3 连续系统的C MEX实现........250
11.3.4 离散系统的C MEX实现........253
11.3.5 混合系统的C MEX实现........256
第12章 控制系统的建模与仿真........259
12.1 离散系统建模........259
12.1.1 离散系统建模的基本概念........259
12.1.2 不同采样速率的彩色显示........260
12.1.3 混合系统建模........261
12.2 经典控制系统的设计与仿真........261
12.2.1 控制系统的时域分析方法........261
12.2.2 控制系统的频域分析方法........262
12.2.3 控制系统的根轨迹分析方法........263
12.2.4 常用控制器的设计与仿真........264
12.3 现代控制系统的设计与仿真........266
12.3.1 现代控制系统的特点........266
12.3.2 模型参考自适应系统........267
12.3.3 实例........267

第 1 章　MATLAB 6.0 概述

SIMULINK 是随 MATLAB 软件一同发行的动态系统通用仿真软件包，使用 SIMULINK 仿真软件必须在 MATLAB 环境下进行。对于 SIMULINK 的初学者来说，第一步必须了解和熟悉 MATLAB 环境的基本使用。因此，本书在开始介绍 SIMULINK 仿真软件的使用之前，首先用两章的篇幅简单介绍 MATLAB 6.0 的基本使用方法，让读者尤其是初学者对 MATLAB 软件以及 SIMULINK 动态系统仿真软件有一个简单的印象。

本章将简要介绍 MATLAB 软件的基本情况，其内容按照循序渐进的次序安排。首先简要介绍了 MATLAB 的发展历史，其次介绍了 MATLAB 6.0 的安装步骤和对系统环境的要求。然后介绍了 MATLAB 6.0 在 5.3 版基础上新增和改进的功能。最后则讲述与 MATLAB 6.0 配套使用的 SIMULINK 4.0 的基本功能和新增特点。读者学习完这一章后，将对 MATLAB 6.0 软件及其 SIMULINK 4.0 的使用有个大致的了解。

1.1　MATLAB 简 介

1.1.1　MATLAB 的发展历程

计算机的出现和发展是现代科学技术发展的巨大成就之一。它对科学技术的几乎所有领域，例如数据处理、统计分析、自动控制以及人工智能等方面产生了极其深远的影响。熟练掌握利用计算机进行科学研究和工程应用的技术，已经成为广大科研设计人员必须具备的基本能力之一。

大部分从事科学研究和工程应用的技术人员常常遇到并为之困扰的是，当我们的计算涉及矩阵运算或画图时，利用 FORTRAN 和 C 等高级编程语言进行程序设计是一件比较麻烦的事情。我们不仅需要对所利用的算法有深刻的了解，还需要熟练掌握程序语言的编程方法。有时这种编程并不是一件容易的事情。例如，当需要计算矩阵的逆矩阵时，我们首先必须选择一个稳定的求逆算法，然后采用某种编程语言经过艰苦繁琐的编程调试工作实现该算法。这使得我们经常将精力花费到这种看起来与我们的工作并无多大关系，且经常是重复性劳动的工作中去。

MATLAB 正是为免除无数上述的局面而产生的。1980 年前后，美国的 Cleve Moler 博士在 New Mexico 大学讲授线性代数课程时，发现应用其他高级语言编程极为不便，便开发了 MATLAB（MATrix LABoratory，即矩阵实验室），这便是采用 FORTRAN 编写的萌芽状态的 MATLAB。经过在该大学进行了几年的试用流传之后，在 Little 的推动下，由 Little，Moler，Steve Bangert 合作，于 1984 年成立了 MathWorks 公司，并推出了该软件的正式版本。从这

时起，MATLAB 的内核采用 C 语言编写，后来的版本又增加了丰富的图形图像处理及多媒体功能，使得 MATLAB 的应用愈来愈广泛。

MATLAB 以商品形式出现以后，短短几年时间，就以其良好的开放性和运行的可靠性，使原来控制领域里的封闭式软件包（如英国的 UMIST，瑞典的 LUND 和 SIMNON，德国的 KEDDC）纷纷淘汰，而改以 MATLAB 为平台重新编写。进入 20 世纪 90 年代，MATLAB 已经成为控制界公认的标准计算软件。

为了准确地将某个控制系统的复杂模型输入给计算机，然后对其进行进一步的分析和仿真，1990 年 MathWorks 公司为 MATLAB 4.x 提供了新的控制系统模型图形输入与仿真工具，并定名为 SIMULAB，该工具很快在控制界得到了广泛的应用。但因此其名字与著名的软件 SIMULA 类似，所以在 1992 年正式改名为 SIMULINK。此软件有两个明显的功能：仿真与连接，即可以通过鼠标在模型窗口中画出所需的控制系统模型，然后利用该软件提供的各种功能对系统进行仿真分析。这种方法使得对一个很复杂的系统的输入变得简单。SIMULINK 的出现，更使得 MATLAB 为控制系统的仿真与其在 CAD 中的应用打开了崭新的局面。

1997 年，MATLAB 5.0 版推出，随后 5.1、5.2、5.3 版相继出现，2001 年初，MATLAB 6.0 问世，MATLAB 也将从此进入一个崭新的时代。经过多年的不断完善和发展，MATLAB 已经拥有更丰富的数据类型和结构、更友善的交互环境、更加快速的图形显示、更广泛的数学函数和工具资源。到了 MATLAB 6.0 版，MATLAB 更是同 Word、Microsoft Visual C++、Java 的主流开发工具和软件实现了无缝连接，通过 MATLAB 自身所包含的各种工具，用户可以轻易实现 MATLAB 语言向 VC++等其他高级语言的互换，并且支持嵌入式系统的开发。MATLAB 已经为专业科技工作人员提供了一个融科学计算、图形显示、文字处理、系统仿真以及系统设计为一体的高效综合的计算环境。

在许多国外的大学和研究所中，MATLAB 已经成为诸如应用代数、数理统计、自动控制、数字信号处理、时间序列分析、动态系统仿真等学科的必修课程之一，在那里攻读学位的本科生、硕士生和博士生在入学之前，都需要花几个月时间熟悉 MATLAB 和相关工具箱的使用，这一做法已经成为许多导师培养学生的基本内容之一。

在国际学术界，MATLAB 也已经被确认为标准的科学计算标准软件。许多权威的学术刊物所刊登的论文，其结果许多也是采用 MATLAB 计算或仿真实现的。而在一些设计部门，MATLAB 被认为是进行高效研究和系统开发的首选工具。很多著名的信号分析软件（如 LabView 等）都以 MATLAB 为主要平台或相互支持。包括 HP 公司的 VXI 硬件等诸多硬件、仪器设备产品都接受 MATLAB 的支持。

1.1.2 MATLAB 6.0 对系统环境的要求

同大多数软件一样，MATLAB 必须具备一定的系统环境才能正常运行。从计算机系统而言，MATLAB 的适应性是很强的，它可以运行在 PC、Macintosh 和 Unix 工作站上。

以下是保证 MATLAB 正常运行的最低配置：

（1）标准配置：

- Pentium，Pentium Pro，Pentium II，Pentium III，或 AMD Athlon 处理器
- 64MB 内存，推荐 128MB
- 显卡要求 8 位以上的彩色显示卡

- 光驱，以安装 MATLAB 之用
- 鼠标
- 至少 250M 以上的硬盘空间（视安装的工具包和帮助文件而定）

（2）可选配置：

- Microsoft Windows 兼容的图形加速卡
- 打印机
- 声卡
- Microsoft Word 7.0（Office 95），8.0（Office 97），或 Office 2000（用于 MATLAB Notebook 的运行）

（3）为创建 MEX 文件，以下任选其一：

- Compaq Visual Fortran 5.0 或 6.1
- Microsoft Visual C/C++ version 5.0 或 6.0
- Borland C/C++ version 5.0，5.02
- Borland C++Builder version 3.0，4.0 或 5.0
- Lcc 2.4（MATLAB 自带的编译工具）

为查阅 PDF 文档，一般还需要 Adobe Acrobat Reader 的支持。

1.1.3 MATLAB 6.0 软件的安装

MATLAB 软件的安装可以选择网络安装和光盘安装两种方式。由于大多数用户是在 PC 机上运行 MATLAB，下面就简单介绍从光盘上安装 MATLAB 6.0 的基本步骤。

第一步：启动安装程序

一般来说，当我们将光盘插入光驱时，计算机将自动启动光盘上的安装程序。否则，我们可以打开操作系统的文件浏览器，在光盘上找到启动文件 setup.exe，双击它即可启动安装。首先出现的是 MathWorks 公司的欢迎画面（如图 1.1 所示）。

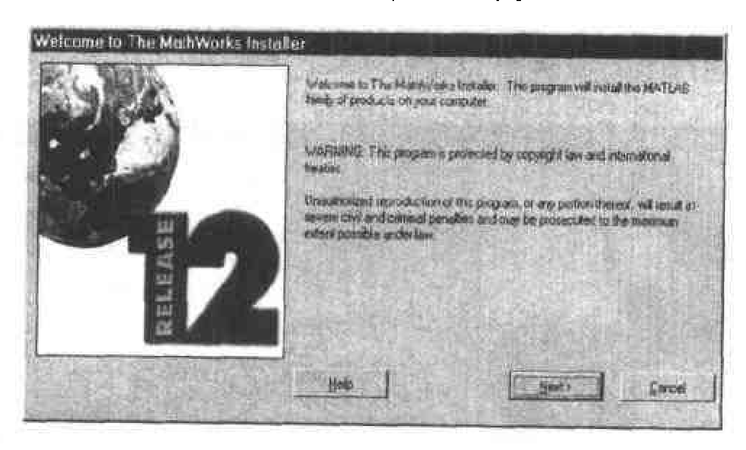


图 1.1 启动安装过程

注意：安装程序需要微软的 Java 虚拟机（Microsoft Java Virtual Machine）支持。如果安装程序在机器中没有检测到 Java 虚拟机，或检测到的版本已经过时，安装程序将提醒用户，并且在安装 MATLAB 之前，首先安装 Java 虚拟机组件。

第二步：授权确认

接下来，安装程序要我们对软件的使用权进行确认，确认的方法是在 PLP（the Personal License Password）栏中输入正确的密码，然后按“Next”按钮。

第三步：选择安装目录及组件

在接下来的对话框中（如图 1.2 所示），需要用户选择 MATLAB 安装的目录（缺省情况下为 C:\matlabR12）和想要安装的组件。以下几点需要注意：

（1）最好将 MATLAB 安装在新的目录下，即使计算机中存在以前的版本。

（2）不要将“private”作为安装目录名，目录名也不要包含空格或以“@”符号开始。

（3）为了安装 SIMULINK 软件包，务必不要忘记勾选 SIMULINK 栏。

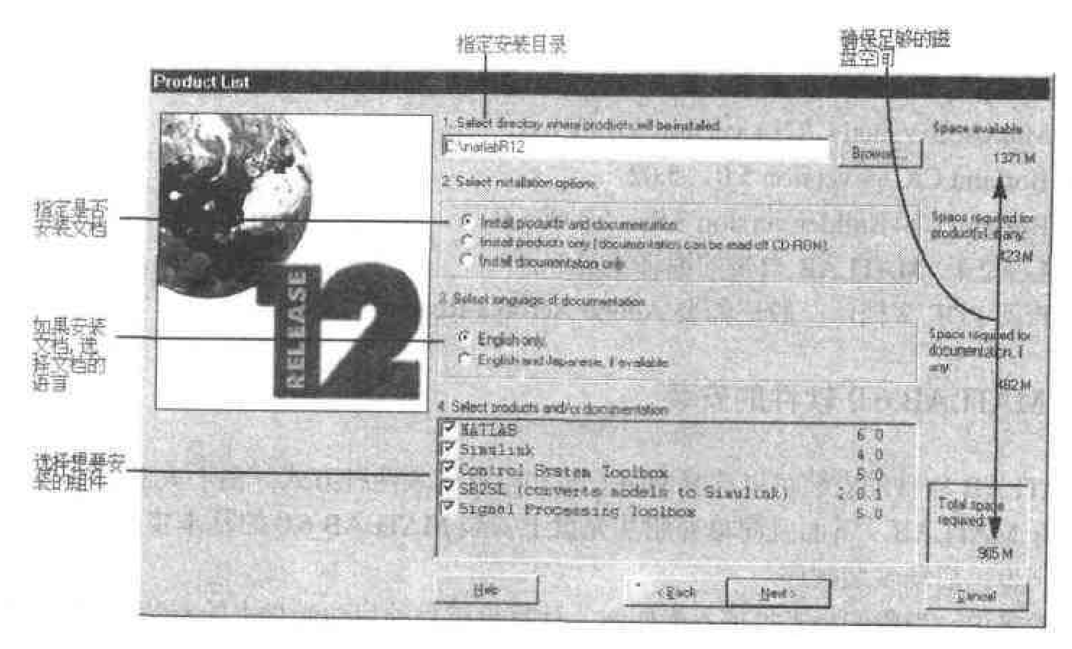


图 1.2　选择安装目录及组件

第四步：安装成功

安装程序将所有的文件拷贝到相应目录下后，出现如图1.3所示的对话框，表明MATLAB已经成功安装。

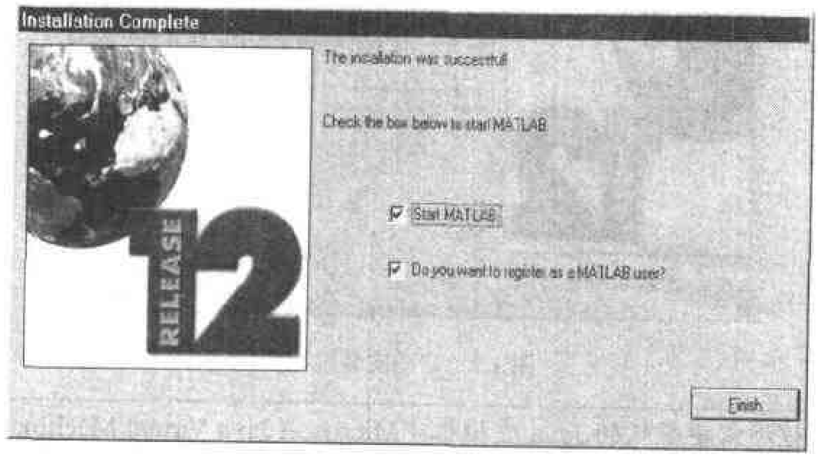


图 1.3　安装过程结束

如果用户安装了需要重新启动计算机的组件，在下面的对话框中，安装程序会提醒用户

是否立即重新启动系统。

1.2 MATLAB 6.0 新特性

1.2.1 MATLAB 语言的发展

MATLAB 语言可以被认为是一种解释性语言，用户可以在 MATLAB 的工作空间中输入一个指令，也可以在编辑器中编写应用程序，应用程序执行时，MATLAB 软件对其中的命令和函数进行翻译，然后在 MATLAB 环境中对它进行处理，最后返回结果。

早期的 MATLAB 语言是由用于矩阵运算的计算机语言发展而来，这也是其名称的由来。它的最基本、最重要的功能就是进行矩阵的运算。但是经过多年的发展，尤其是 MathWorks 公司密切注意科技发展新动向，与不同领域的全球知名专家合作，及时推出新的工具箱，使得 MATLAB 成为跨学科、高度综合、专业化程度很高的计算与仿真工具。因此，MATLAB 有“巨人肩上的工具”之称。MATLAB 已经从最初的矩阵计算工具渗透到科学与工程计算的多个领域，在自动控制、信号处理、神经网络、模糊逻辑、小波分析、图像处理等多个方向都有广泛的应用。

MATLAB 从 5.3 版起已经成为比较成熟的产品。其标志是数据结构的完善和面向对象的编程技术的运用。若干工具箱的升级，使应用工具包表现更加友好，而且功能大大增强。

1.2.2 MATLAB 6.0 的开发环境

早期的 MATLAB 工作环境比较单调，5.x 版本改变了过去单调依靠在指令窗口输入指令进行操作的单一方式。5.3 版将图形显示窗口改造成为交互操作的可编程图形界面。而 MATLAB 6.0 在 5.3 版本的基础上对图形界面进行了进一步的改进。

1. MATLAB 桌面

MATLAB 6.0 以前的版本在启动后，首先出现的是 MATLAB 的命令窗口。而 MATLAB 6.0 版在用户界面上发生了较大的变化，出现了所谓的 MATLAB 桌面的集成开发环境。该环境类似于 Microsoft 公司的 Visual Studio 开发环境，用户可以在统一的环境中完成对文件、变量和 MATLAB 程序的管理。并且桌面的设置可以依照个人的喜好进行定制。MATLAB 启动后，首先将进入 MATLAB 桌面环境，如图 2.1 所示。

2. 更快的启动时间

对于运行在网络服务器上的 MATLAB，其启动时由于需要搜索网络中所有客户端的工具箱的路径信息，往往会导致过长的等待时间。MATLAB 6.0 新增的工具箱路径缓存技术可以有效避免这种情况的发生。它将所有工具箱的路径信息缓存到本地服务器的根目录下，启动时将这些信息读取进来，而不需要访问其他的客户端，从而提高了系统启动时间。

3. 输入向导

在 MATLAB 6.0 中，往工作空间中添加新 ASCII 文本或变量，最简单的方法是使用新的

输入向导（Import wizard）。使用方法是：

在命令窗口中选择“File”，单击“Import Data”菜单，输入向导将弹出一个新的文件选择对话框，在该对话框中，我们可以选择将要导入工作空间的数据文件。输入向导打开数据文件并显示文件所包含的所有变量，我们可以选择其中需要导入工作空间的变量，然后按下“Finish”按钮。输入向导可以对其中包括图像、声音在内的各种数据格式进行自动识别。

4. 更多的系统函数

增加了新的系统函数，并对部分的系统函数进行了修改，方便了用户的使用。

1.2.3 MATLAB 的数学计算能力

1. 矩阵计算

新增的矩阵运算库大大提高了 MATLAB 原有的矩阵运算能力，并进一步提高了许多算法的执行速度。这种基于 LAPACK 的运算库采用 Fortran 语言编写，包含大量的子程序。新增了许多函数可以解决更广泛的数学问题。

2. 微分方程解算器

新增的解算器扩展了 MATLAB 原有的求解微分方程的能力。例如新增的 bvp4c 和 pdepe 可以分别求解两点边值问题和初始边界问题。

3. 更快的傅利叶变换

依靠麻省理工学院的 FFTW 库，MATLAB 6.0 可以使傅利叶变换的速度进一步提高。

4. 数据统计界面

MATLAB 6.0 新增了一个数据统计界面来完成：

- 计算一个 graph 窗口中所画点的统计信息。
- 将统计信息导入工作空间。

在一个 graph 窗口中将这些统计信息显示出来。

选择菜单“Tool”中的“Data Statistics”，将出现如图 1.4 所示的对话框。

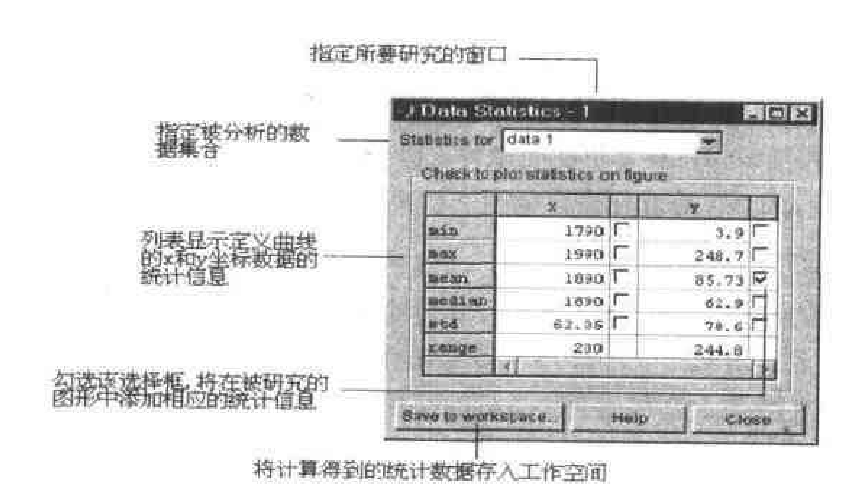


图 1.4　图形窗口的数据统计对话框

其中显示了某个 graph 窗口中所画点的统计信息。可以将统计信息画出来（选中相应的检查框），也可以将它导入工作空间。

增加了新的系统函数，并对部分的系统函数进行了修改，方便了用户的使用。

1.2.4 MATLAB 6.0 的其他新特点

1. 编程和数据类型

MATLAB 6.0 新增了对 Java 的接口。用户可以从某个 Java 类中创建对象，并调用该对象中的方法。用户可以使用一个已经存在的 Java 类，也可以根据需要自己定义 Java 类。

MATLAB 6.0 还增加了与 Java 交互的 API，通过这些接口函数，用户可以在 MATLAB 环境中方便地使用 Java 类。

MATLAB 6.0 增加了串口 I/O 的通信函数，使用户能够方便与挂接在串门中的调制解调器等设备进行通信。

MATLAB 5.3 具备六种不同的基本数据类型，即双精度数组、字符串数组、元胞数组、构架数组、稀疏矩阵和 unit8 数据。而 MATLAB 6.0 新增了函数句柄（function handle）的数据类型。在 MATLAB 6.0 中，用户可以创建任何函数的句柄，在进行函数调用时可以用这个句柄代替原来的函数本身。通过这个类似于 C 语言函数指针的数据类型，用户可以很容易实现不同输入参数的函数之间的重载。

2. 图形显示

MATLAB 6.0 新增了一个称为属性编辑器的图形用户界面，通过它可以方便地修改 MATLAB 窗口中的图形对象。修改的内容包括图形所在的窗口、坐标、光源、线的宽度、颜色、标注等等，避免了旧的版本中从指令窗口修改图形属性的麻烦。

有几种不同的方法启动属性编辑器。

如果窗口在允许对象编辑的状态下，可以选中某个图形对象，然后单击鼠标右键，弹出上下文菜单，选择其中的“Properties”；或者直接双击需要编辑的图形对象（如果是文本，则只弹出文本编辑框）。

如果窗口在禁止对象编辑的状态下，可以选择“Edit”中的“Figure Properties”，“Axes Properties”，或“Current Object”，这时窗口自动切换到允许对象编辑的状态。

当然，也可以在命令窗口中键入propedit指令来启动属性编辑器。

众所周知，MATLAB 之所以吸引越来越多研究人员的关注和喜爱，一个重要的原因是 MATLAB 在进行科学计算的同时，提供了丰富的画图函数，使得我们能够方便的对数据进行图形显示，让结果直观地显示在我们面前。MATLAB 6.0 在绘图方面则得到了进一步的增强，新增了大量的三维绘图函数，使数据的图形显示更加丰富，对于三维图形，新增的透明属性让人们更加领略到 MATLAB 带给用户的方便。

3. 图形用户界面设计（GUI）

MATLAB 从 5.x 起增加了图形用户界面设计（GUI）的环境，通过它，用户可以方便地绘制交互性能优良的应用程序。MATLAB 6.0 在 5.3 版本的基础上对 GUI 进行了重新设计，使图形用户界面的开发更加方便，并且采用一种新的 FIG 文件取代原来的 m 文件保存图形界面的相关信息。创建的图形用户界面既支持单进程也可以支持多进程。

1.3　SIMULINK 4.0 概述

1.3.1　什么是 SIMULINK

SIMULINK 是 MathWorks 公司开发的一个有重要影响力的软件产品。它的前身 SIMULAB 产生于 20 世纪 90 年代初，以工具库的形式挂接在 MATLAB 5.3 上。后改名为 SIMULINK。SIMULINK 不能独立运行，只能在 MATLAB 环境中运行。其版本为：与 MATLAB 5.2 一同发行的 SIMULINK 2.2；与 MATLAB 5.3 一同发行的 SIMULINK 3.0；MATLAB 6.0 则包含 SIMULINK 4.0 版本。

SIMULINK 是一个用来对动态系统进行建模、仿真和分析的软件包，它支持连续、离散或两者混合的线性和非线性系统，也支持具有多种采样速率的多速率系统。

SIMULINK 为用户提供了用方框图进行建模的图形接口，采用这种结构画模型就像用笔和纸来画一样容易。与传统的仿真软件包用微分方程和差分方程建模相比，具有更直观、方便、灵活的优点。SIMULINK 包含 Sinks（输出方式）、Source（输入源）、Continuous（连续环节）、Nonlinear（非线性）、Discrete（离散环节）、Signals & System（信号与系统）、Math（数学模块）和 Functions & Tables（函数和查询表）子模型库。随着该软件的发展，子模型库不断得到丰富和发展。用户也可以定制和创建用户自己的模块。

用 SIMULINK 创建的模型可以具有递阶结构，因此用户可以采用从上到下或从下到上的结构创建模型。用户可以从最高级开始观察模型，然后用鼠标双击其中的子系统模型，来检查下一级的内容，以次类推，从而可以看到整个模型的细节，帮助用户理解模型的结构和各模块之间的相互关系。

在模型创建以后，用户可以通过 SIMULINK 的菜单命令或 MATLAB 的命令窗口输入命令来对它进行仿真。通过 Scope 模块和其他的画图模块，在仿真进行的同时，可以观察仿真结果。除此之外，用户还可以在改变参数来迅速观察系统中发生的变化。仿真的结构可以同时存放到 MATLAB 的工作空间内，供以后的计算、分析之用。

此外，SIMULINK 还包括线性化和平衡点分析等模型分析工具。MATLAB 自身所带的所有应用工具箱，同样适用于 SIMULINK 环境中。这就大大扩展了 SIMULINK 的适用范围，也使 SIMULINK 进入到许多专业领域，成为科学计算和动态系统仿真的有利工具。由于 MATLAB 和 SIMULINK 是集成在一起的，因此用户可以在这两种环境中对自己的模型进行仿真、分析和修改。

1.3.2　SIMULINK 4.0 的组成

SIMULINK 4.0 是由模块库、模型构造及指令分析、演示程序等几部分组成。经过几年的发展，SIMULINK 4.0 已经发展成为一个系列化的产品。例如，结合 Stateflow，SIMULINK 可以对更加广泛意义上的具有时间触发和离散事件系统进行分析和仿真；如果结合 Real-Time Workshop，用户可以进行实时的系统仿真，能够将 SIMULINK 中的仿真框图转换成 C 语言

等其他高级语言，并可直接运行于相应的硬件上；如果结合 DSP Blockset，它还可以进行 DSP 装置和系统的快速设计和仿真。另外，SIMULINK 如果在诸如 Communication Toolbox，Nonlinear Control Design Blockset，Power System Blockset 等工具箱的配合下，还可以完成对诸如通行系统、非线性控制系统、电力系统的建模、分析和仿真。因此在许多辅助工具的帮助下，SIMULINK 不再是单一的动态系统仿真软件，它的作用和应用领域已经得到进一步的扩展。

1.3.3 SIMULINK 4.0 的特点

1. 库浏览器的变化

SIMULINK 3.0 以前的标准模型库是子库分层的块图结构，到 SIMULINK 3.0 发展成为树状的文件夹形式，SIMULINK 4.0 则是将模型库在一个预览面板中以图标形式显示出来。用鼠标将模块从预览面板中拖到模型窗口中，就可以生成该模块的一个实例。

2. 对矩阵信号的支持

旧版的 SIMULINK 只支持标量或矢量信号，而 SIMULINK 4.0 的许多模块提供了对矩阵信号的支持，进一步增加了系统仿真的灵活性。SIMULINK 3.0 提供了对模型方框图的调试工具。然而这种调试是比较初步的。而 SIMULINK 4.0 在 3.0 的基础上提供了一个新的专门用于系统调试的图形用户界面（GUI），从而使得对模型方框图的跟踪调试更加方便。

另外，众多工具箱的升级也使 SIMULINK 的功能进一步增强，使用更加方便。

第 2 章　熟悉 MATLAB 6.0 环境

本章简要介绍了 MATLAB 的桌面环境、基本操作和用法，作为入门旨在使读者对 MATLAB 软件的基本使用有个大致的了解。

MATLAB 软件本身博大精深，内容十分丰富。它不仅包含诸如 SIMULINK 等众多的软件工具包，而且具有丰富的数学计算功能和代表目前世界上大多数主要学科领域的专业工具箱。因此要在一个章讲清所有的用法是不可能的，这也不在本书内容的范围之内。我们主要介绍 MATLAB 最基本、也是最常用的一些用法，而不涉及各种工具箱的使用。读者学习完这一章后，将顺利跨过 MATLAB 学习的门槛，为以后学习 SIMULINK 的使用打下基础。熟悉 MATLAB 的读者可以跨过这一章，直接下面的学习。

2.1　MATLAB 6.0 的桌面环境

启动 MATLAB 的可执行文件是 matlab.exe，该文件放在 matlab\bin 目录下。一般完成 MATLAB 的安装后，安装程序会自动在桌面上增加一个快捷方式图标，用鼠标双击这个图标，就会自动创建如图 2.1 所示的 MATLAB 6.0 的桌面环境。当然，也可以在 Windows 的“开始”菜单中寻找“matlab”的菜单项，单击即可启动 MATLAB 6.0 的桌面环境。

由于 MATLAB 6.0 在用户界面上不同于以往的版本，用专门的桌面化环境代替了原来单一的命令窗口，下面我们将详细介绍 MATLAB 6.0 的桌面环境。

MATLAB 6.0 启动后将出现如图 2.1 所示的工作环境。

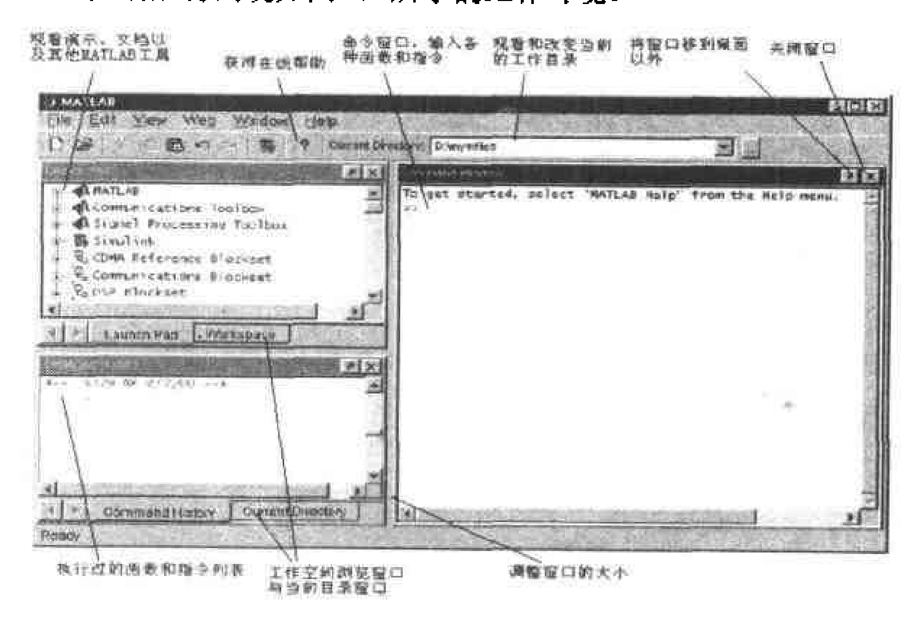


图 2.1　MATLAB 的桌面环境

各部分的功能说明如下：

命令窗口（Command Window）	运行MATLAB函数
命令历史（Command History）*	记录以前在命令窗口中输入的指令，用户可以将其中需要重新执行的指令拷贝到命令窗口。
快捷栏（Launch Pad）*	列举MathWorks公司的所有产品和相关文档，用户可以通过鼠标选择需要运行的程序、浏览帮助文档和观看演示。
帮助浏览器（Help browser）	取代了旧版的帮助桌面（Help Desk）。
当前目录浏览器（Current Directory browser）*	浏览MATLAB及相关文件。完成文件操作。例如打开文件、在文件中查找字符串等等。
工作空间浏览器（Workspace browser）	浏览或修改工作空间中的数据。
数组编辑器（Array Editor）	以表的形式显示数组的内容，并且可以修改其中的某几项。
编辑/调试器（Editor/Debugger）	对M文件进行创建、编辑和调试，功能更加强大。

需要注意的是，并不是所有的工具都是MATLAB 6.0一启动就出现的。以*标注的表示这是MATLAB 6.0新增的工具。

2.2 MATLAB基本指令和用法

2.2.1 数值、变量和表达式

MATLAB数值一般采用十进制数表示，如：

45　　-103　　3.56　　2.34e8　　2.2e-1

在采用IEEE浮点算法的计算机上，数值的相对精度为eps，大约保留16位有效数字。数值范围为1×10^{-309}~1×10^{309}之间。

在MATLAB中，变量名、函数名是对字母大小写敏感的，变量名的第一个字符必须是英文字母，最多可以包含31个字符，变量名中不能包含标点、空格字符。

在MATLAB中存在一些固定的变量，如eps、pi、Inf、NaN、realmax、realmin等。变量eps为机器零阈值，在决定诸如矩阵奇异时，可作为容许误差。另外用户也可将它作为包括零值的其他任务值。Inf表示正无穷大，例如当除数为零时，其结果为Inf。

从这里可以看出，在MATLAB中，被零除是允许的，它不会导致程序的中断，这种特殊的表达方法将会在以后的计算中发挥作用。

MATLAB中的表达式遵循我们日常中的习惯写法。

MATLAB还支持复数的使用，虚数符号为i或j。

2.2.2　向量运算

向量的使用和运算始终是 MATLAB 的核心内容。MATLAB 具有丰富的数学函数处理向量或矩阵的运算。

1. 构造向量

在 MATLAB 中，可以用“：”产生不同的向量，如 x=1:4 产生一个 1~4 单位增量的行向量。

```
>>x=1:4
ans =
1  2  3  4
```

注意：指令行前面的“>>”表示该行是由用户输入的，否则表示输入指令的计算结果。本书以后的内容都遵循这样的约定。

也可以产生单位增量不等于 1 的行向量，方法是把增量放在起止量的中间，并用冒号隔开。

```
>>y = 0:pi/4:pi
y =
0      0.7854     1.5708     2.3562     3.1416
```

符号“:”也可以用来产生矩阵的表格。为了产生纵向矩阵形式，首先用它产生行向量，然后进行转置，再利用所得的列向量计算出另一列向量，即可合成有两列的矩阵，如

```
>>x = (0: 0.2: 3.0);
>>exp(-x).*sin(x);
>>[x y]
ans =
Columns 1 through 7
     0      0.2000     0.4000     0.6000     0.8000     1.0000     1.2000
Columns 8 through 14
1.4000     1.6000     1.8000     2.0000     2.2000     2.4000     2.6000
Columns 15 through 21
2.8000     3.0000          0     0.7854     1.5708     2.3562     3.14162.
```

2. 向量的下标

MATLAB 的下标具有很重要的功能，可以在对矩阵的行、列子矩阵处理时使用，也可以用来产生向量。使用下标和向量，会使运算更为清晰和方便。单个的矩阵元素可以在括号中用下标直接表达。例如 A(3,3)为 A 矩阵的第三行第三列的元素；A(3,1)表示 A 矩阵第三行第一列的元素。

下标可以是一个向量，例如假设 x 和 v 都是向量，则 x(v)也是一个向量：[x(v(1)) x(v(2)) … x(v(n))]。对于矩阵而言，向量下标可以将矩阵中邻近的元素构成一新的子矩阵。假设 A 是一个 10×10 的矩阵，则 A(1:5,3)指 A 中由前 5 行对应第三列元素组成的 5×1 子矩阵。又如 A(1:5,7:10)是前 5 行对应最后 4 列组成的 5×4 子矩阵。

使用 “:” 代替下标，可以表示所有的行或列。如 A(:, 3)表示第三列元素组成的子矩阵，A(1:5, :)代表由前 5 行所有元素组成的子矩阵。

对于子矩阵的赋值语句，“:” 符号具有更明显的优势。如

```
>>A(: , [3, 5, 10]) = B(: , 1:3)
```

表示将 B 矩阵的前三列，赋值给 A 矩阵的第三、第五和第十列。

通常如果 v 和 w 是具有整数性质的向量，则 A(v,w)通过取出行下标 v 和列下标 w 对应的 A 的元素而形成新的矩阵。于是，A(:,n:-1:1)表示由原来 A 矩阵中取 n 至 1 负增长的列的元素组成一个新的矩阵，其行数仍为原来 A 矩阵的行数，列数为 n。

如果 A(:)在赋值语句的右边，表示将 A 的所有元素在一个长的列向量中展成串，如

```
>>A = [1 2; 3 4; 5 6], b = A(:)
A =
1	2
3	4
5	6
b =
 1
 3
 5
 2
 4
 6
```

在赋值语句左边 A(:)可以重新组成与刚才的 A 具有相同阶数的矩阵，这相当于在原来的 A 没有被清除的情况下，用新的元素进行置换，实际上起着一种提供格式的作用。例如：

```
>>A(:) = 11:16
A =
11	14
12	15
13	16
```

这时 A 矩阵已经被新的元素取代了，6 个新的元素取代了原来矩阵中的元素，重新组成 3×2 矩阵。

2.2.3　矩阵的简单运算

1. 空矩阵

矩阵 x=[]将产生一个 0×0 的矩阵，使用这个矩阵不会引起错误。x=[]与 clear 指令不同，后者是将 x 清楚出工作空间，而空矩阵仍旧存在于工作空间，只是其大小为 0。我们可以采用指令 exist 测出其确实存在。

如果给出空矩阵，确定的矩阵函数，如 det( )，cond( )，prod( )，sum( )等会返回特定值。例如，当我们给定空矩阵时，prod( )、det( )和 sum( )将分别返回 1、1 和 0。

空矩阵虽然在数学上是空的，但是通常在我们不知道定义某个矩阵的确切大小时，将该矩阵定义为空矩阵。

2. 矩阵的转置

矩阵的转置通过“ ′ ”来表示和实现。例如：

```
>>A = [1 2 3;4 5 6;7 8 9], B = A′
A =
     1     2     3
     4     5     6
     7     8     9
B =
     1     4     7
     2     5     8
     3     6     9
```

也可以直接对向量进行转置运算，如：

```
>>B = [-1 0 2] ′
B =
 -1
 0
 2
```

如果 z 是复数矩阵，则 z′ 为它们的复数共轭转置。非共轭转置使用 z.′ 或 conj(z′)求得。

3. 矩阵的乘法

矩阵乘法用“*”表示。这与数学上的形式是一致的。两个相同维数向量的内积（即数学上的点积）也可以用这种方法实现。例如：

```
>>x = [-1   0   2]′ ; y = [-2  -1   1]′ ; x′ *y
ans =
 4
```

在 MATLAB 中还可以进行矩阵和标量相乘，标量可以是乘数也可以是被乘数。矩阵和标量相乘是进行矩阵中的每个元素都与此标量相乘的运算。

4. 矩阵的除法

在 MATLAB 中用两种不同的矩阵除法符号“\”和“/”分别表示左除和右除。如果 A 矩阵是非奇异的，则 A\B 和 A/B 都可以实现。A/B 相当于 A 的逆左乘 B 矩阵，也就是 inv(A)*B，而 B/A 等效于 A 矩阵的逆右乘 B 矩阵。

通常，X = A\B 是 A*X = B 的解，X = A/B 是 X*A = B 的解。一般情况下，A\B 不等于 A/B。

5. 矩阵的乘方

A^P 表示 A 的 P 次方，如果 A 是一个方阵，P 是一个标量，且 P 是大于 1 的整数，则 A 的 P 次幂为 A 自乘 P 次。如果 P 不是整数，则计算涉及特征值和特征向量的问题。例如，假设[V D]=eig(A)，则：

A^P = V*D.^P/V

其中，V 和 D 分别为矩阵 A 的特征向量矩阵和特征值矩阵。

如果 P 是矩阵而 A 是标量，以及 A、P 都是矩阵，则 A^P 都是不合法的。

6. 创建矩阵的函数

在 MATLAB 中有很多的函数用于创建某些特殊的矩阵，例如：

- eye(size(A))产生与 A 矩阵同阶的单位矩阵；
- zeros 和 ones 产生 0 和 1 矩阵；
- rand 产生随机元素的矩阵；
- 函数 diag( )、triu( )和 tril( )分别创建对角、上三角和下三角矩阵；
- 函数 size( )显示一个包含两个元素的向量：矩阵的行与列的个数。函数 length( )返回向量的长度或矩阵行数和列数中的最大值。

2.2.4 矩阵的特殊运算

MATLAB 的数学处理能力很大一部分是由它的矩阵函数扩展而来的。MATLAB 6.0 为用户提供了强大的矩阵运算函数，这里只介绍其中最基本的一些内容。

1. 矩阵的三角分解

矩阵的三角分解是指将一个方阵表示成一个上三角阵与一个下三角阵的乘积。这种分解方法又称为“LU”分解。这里使用的算法是高斯消去法。分解的因数从 lu 函数中得到，利用这种分解，求逆可得到逆矩阵，求 det 可得到行列式的值。它是求线性方程组的基础，也是方阵左除和右除的基础。例如：

```
>> A = [1 2 3;  4 5 6;  7 8 9];
>>[L U] = lu(A)
L =
     0.1429    1.0000         0
     0.5714    0.5000    1.0000
     1.0000         0         0
U =
     7.0000    8.0000    9.0000
          0    0.8571    1.7143
      0         0    0.0000
```

还可以验证 LU 分解的正确性：

```
>>L*U
ans =
     1     2     3
     4     5     6
     7     8     9
```

2. 矩阵的正交变换

矩阵的“QR”分解表示为正交矩阵和上三角矩阵的乘积。例如：

```
>> A = [1 2 3;4 5 6;7 8 9;10 11 12];
>>[Q R] = qr(A)
Q =
   -0.0776   -0.8331   -0.2636   -0.4801
   -0.3105   -0.4512    0.7093    0.4437
   -0.5433   -0.0694   -0.6278    0.5530
   -0.7762    0.3124    0.1821   -0.5166
```

```
R =
   -12.8841   -14.5916   -16.2992
         0    -1.0413    -2.0826
         0          0    -0.0000
         0          0          0
```

可以验证 Q*R 的值就是原矩阵 A。观察 R 矩阵的三角结构，其对角线的下半部分全是零，在对角线上 R(3,3)元素为零。说明 R 与原来 A 矩阵不是满秩的，QR 分解可以解决方程数多于未知数的线性系统问题：

3. 奇异值分解

奇异值分解的调用函数为[U,S,V] = svd(A)。

在奇异值分解中产生三个因数矩阵 U、S 和 V，满足：A = U*S*V′。

U 矩阵和 V 矩阵是正交矩阵，S 矩阵是对角矩阵，而且 svd(A)恰好返回 S 的对角元素，就是 A 的奇异值。

其他几种函数也可以进行奇异值分解，这包括广义逆矩阵 pivn(A)，秩 rank(A)，矩阵范数 norm(A,2)和条件数 cond(A)。

4. 矩阵的特征值

如果 A 是 n×n 矩阵，称满足 $Ax = \lambda x$ 的 λ 值为 A 的特征值，x 为 A 的特征向量。计算特征值使用函数 eig(A)，以列向量形式返回特征值。如果 A 是实对称矩阵，特征值为实数，如果不对称，特征值常为复数，例如：

```
>> A = [0 1;-1 0 ];
>>eig(A)
 ans =
     0 + 1.0000i
     0 - 1.0000I
```

求解特征值和特征向量可以用双赋值语句得到：

[X，D] = eig(A)

D 的对角元素是特征值，X 为矩阵，它的列是相应的特征值，以使得 A*X = X*D。

2.2.5　元胞数组

在 MATLAB 尤其是 SIMULINK 环境中，常常用元胞数组（Cell array）来定义模块的参数，它也是 MATLAB 中的一个比较重要的数据类型。

元胞数组的组成元素称为元胞（Cell）。每个元胞在数组中是平等的，它们只以下标区分。元胞可以存放任何类型、任何大小的数组（如任意维数值数组、字符串数组、符号对象等）。而且，同一个元胞数组中各元胞中的内容可以不同。

与数值数组一样，元胞数组维数不受限制，可以是 1 维、2 维或更高维，不过 1 维数组使用最多。元胞数组对元胞的编址方法有单下标编址和全下标编址两种。

以 3 维元胞数组为例，全下标编址由 3 个序号组成。编址中的第 1 序号是“行”号，第 2 个是“列”号，第 3 个是“页”号。而单下标编址中，第 1 行第 1 列第 1 页的元胞是序号为 1，然后沿第 1 列往下记号 2，3，4 等等，直到第一列的元胞全部数完；接下来数第 2 列

的第 1 行元胞，然后再沿第 2 列往下数，依次类推，直到这 1 页的元胞全部数完。紧接着往下的是第 2 页上的第 1 行第 1 列元胞，再沿列而下，如此进行直到全部数完。

我们知道，在字符数组或数值数组中，由于同一数组的数据类型都相同，因此对元素的寻址也是直接的。例如，对于 2 维数组来说，A(2,3)就表示数组 A 的第 2 行第 3 列上的元素。

对元胞数组来说，元胞和元胞里的内容是两个不同范畴的东西。因此，访问元胞本身与访问元胞中的内容是属于两种不同的操作。

以 2 维元胞数组为例，A(2,3)是指元胞数组中的第 2 行第 3 列的元胞元素，而 A{2, 3}是指元胞数组中的第 2 行第 3 列的元胞中所存储的内容。这两者的区别仅在于所使用的括号不同。前者所用的是圆括号，而后者采用的是大括号。下面的例子将演示如何创建元胞数组并显示其中的内容。

例2.1：元胞数组的创建

```
>>string = char('某个字符串形式');        %产生字符串
>> R = reshape(1:9, 3, 3);               %产生 3×3 实数矩阵
>>C = [1+2i];                            %产生一个复数
>>S = sym('sin(-3*t)*exp(-t) ');         %产生数学表达式
>>A(1, 1) = {string}; A(1, 2) = {R}; A(2,1) = {C}; A(2, 2) = {S};
>>A                                      %显示元胞数组
A =
    '某个字符串形式'          [3x3 double]
[1.0000+ 2.0000i]          [1x1 sym     ]
>>B{1, 1} = string; B{1, 2} = R; B{2,1} = C; B{2, 2} = S;
>>celldisp(B)                            %显示元胞数组内容
B{1,1} =
某个字符串形式
B{2,1} =
    1.0000 + 2.0000i
B{1,2} =
       1      4      7
       2      5      8
       3      6      9
B{2.2} =
sin(-3*t)*exp(-t)
```

例2.2：元胞数组内容的调用

接上例

（1）取一个元胞

```
>>f1 = A(2, 2)
class(f1)
f1=
     [1x1 sym]
ans =
cell
```

（2）取一个元胞的内容

```
>>f2 = A{2,2}
>>class(f2)
```

```
f2 =
sin(-3*t)*exp(-t)
ans =
sym
```

（3）取元胞内的子数组

```
>>f3 = A{1,2}(:, 1)
f3 =
     1
     2
     3
```

（4）同时访问多个元胞数组的内容

```
>>[f4, f5, f6] = deal(A { [ 1, 3, 4 ] } )        %取三个元胞元素内容，分别赋值给三个变量
f4 =
某个字符串形式
f5 =
     1     4     7
     2     5     8
     3     6     9
f6 =
sin(-3*t)*exp(-t)
```

2.2.6 结构数组

同元胞数组一样，结构数组（structure array）也能在一个数组里存放各类数据。但从某种意义上讲，结构数组比元胞数组具有更强的灵活性。

结构数组的基本组成成分是结构（structure）。数组中的每一个结构都是平等的，它们之间通过下标进行区分。结构必须在划分“域”后才能使用。数据不能直接存放在结构中，而只能存放在相应的域中。结构中的域可以存放任何类型和大小的数组。而且，不同结构的同名域中存放的内容可以不同。

与数值一样，结构数组的维数不受限制，可以是 1 维、2 维或更高维，不过 1 维结构数组用得比较多。

由于结构数组中具有多个域的结构，因此，在访问结构数组域中的存储内容时必须加上结构的域名，如 A(1,2).f 标示 A 结构数组中的第 1 行第 2 列中名为 f 的域中所存放的内容。

通过 cell2struct 和 struct2cell 指令可以实现元胞数组与结构数值之间的转换。下面通过例子来说明如何创建和使用结构数组。

例2.3 结构数组的创建

```
>>student.name = '张三';                    %直接对结构的各个域进行赋值
>>student.year = 18;
>>student.score = [89 94 93 100 88.5 99];
>>student.address = '朝阳大街 102 号';
>>student                    %显示结构数组的内容，这里结构数组只包含一个结构。
student =
      name: '张三'
      year: 18
```

```
        score: [89 94 93 100 88.5000 99]
            address: '朝阳大街 102 号'
>>student.score                         %显示 score 域中的内容
ans =
   89.0000    94.0000    93.0000   100.0000    88.5000    99.0000
```

例2.4：多维结构数组的创建与显示

```
>>student(2, 3).name  =  '李四';          %创建 2×3 结构数组
>>student                               %显示结构数组的结构
student =
2x3 struct array with fields:
    name
    year
    score
    address
>>student(2, 3)                        %显示相应的结构
ans =
name: '李四'
 year: []
score: []
  address: []
```

除了直接对域进行赋值的方法外，还可以采用 struct 的构造函数来创建结构数组。不过，该指令不可以直接创建带子域的结构数组。

例2.5：采用构造函数来创建结构数组

```
>>a = cell(2, 3);    %创建2×3的空元胞数组
>>student = struct ('name', a, 'year', a, 'score', a, 'address', a)
student =
2x3 struct array with fields:
name
year
score
address
```

由于结构数组中的域是存放数据的地方，因此访问或存取其中的内容必须预先知道域的名字。可以通过 findnames 指令来获取结构数组的域名，然后通过 getfield 和 setfield 指令来实现结构数组中数据的存取。其使用方法是：

```
FN = findnames（S_n）                                    %获得结构域名
FC = getfield（S_n，{S_index}, f_name, {f_index}）      %获取具体结构域中的内容
S_n = setfield(S_n，{S_index}, f_name, {f_index}, value)%设置具体结构域中的内容
```

参数说明：

fieldnames：　函数输出一维元胞数组 FN，它的每个元胞是被 S_n 的每一个域赋值。

Getfield:　函数输出的 FC 是具体结构域中的内容。

Setfield：　函数输出的是结构数组本身，但是它的某些域中的内容被修改。

S_n：　可以是结构数组名。

{S_index}：　当 S_n 为结构数组时，S_index 指定元素结构的下标。
{S_index}必须是元胞数组形式。

f_name:　　　指定的域名。必须是字符串。
{ f_index }:　　指定域中数组的下标。{ f_index }必须是元胞数组的形式。
Value:　　　具体的设定值。

2.2.7　数据的图形显示

数据的可视化是 MATLAB 的主要组成部分。MATLAB 中的图形指令具有很强的功能，可以实现各种图形的 1 维或 2 维显示，可以修改其中的某些图形对象属性，甚至可以改变图形的视角。由于 MATLAB 6.0 中的图形函数十分丰富，作者不可能一一介绍，这里只是通过几个例子来熟悉一下几个常用指令的用法。

例2.6：二维图形的绘制

```
>>x = linspace(0, 2*pi, 30);
>>y = sin(x);
>>z = cos(x);
>>plot(x, y, '-', x, z)                    %绘制二维曲线
>>grid
>>xlabel('X')                               %输出X轴说明
>> ylabel('Y')                              %输出Y轴说明
>>title('sine and cosine curve')            %在图形上方加上图形说明
>>text(2.5, 0.7, 'sin(x) ')                 %在适当位置为图形加上注释
```

运行的结构如图2.2所示。

例2.7：　三维图形的绘制

（1）三维曲线的绘制

```
>>t = 0: pi/50: 10*pi;
>>plot3(sin(t), cos(t),t )                  %绘制三维曲线，结果如图2.3所示。
>>grid
```

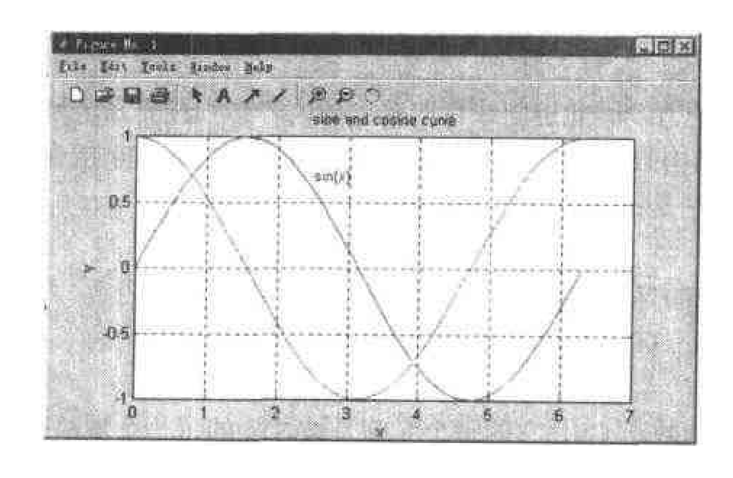


图 2.2　二维图形的绘制

图 2.3　三维曲线的绘制

（2）三维曲面的绘制

```
>>x = -7.5: 0.5: 7.5;
>>y = x;
```

```
>>[X, Y] = meshgrid(x,y);
>>R = sqrt(X. ^2+Y. ^2)+ eps;
>>Z = sin(R). /R;
>>mesh(X, Y, Z)                    %绘制三维曲面，如图2.4所示。
```

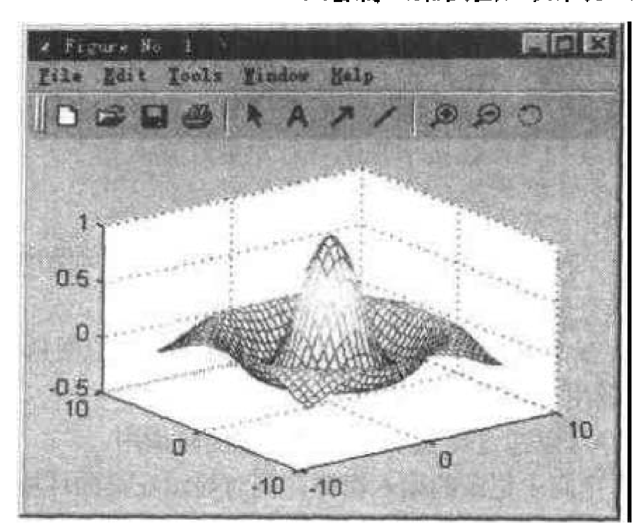


图 2.4　三维曲面的绘制

2.2.8　命令窗口基本指令

MATLAB 6.0 为用户的操作提供了丰富的图形界面，常用的指令功能都可以通过相应的菜单或工具按钮来完成。但对于许多老版本的用户，他们更习惯于通过指令方式来实现。用户通过在 MATLAB 桌面上的指令窗口中输入适当的指令，可以完成几乎所有的计算和设计工作，例如前几小节的例子我们都是通过指令来完成的。下面我们介绍一些常用的指令及其用法，使读者了解 MATLAB 指令的基本使用。

1. 常用控制指令

cd	设置当前工作目录
clf	清除图形窗口
clc	清除命令窗口中的显示内容
clear	清除 MATLAB 工作空间中所有或指定的变量
dir	列出指定目录下的文件和子目录清单
edit	打开 M 文件编辑器
exit（quit）	关闭（退出）MATLAB
more	在命令窗口中分页显示输出的内容
pack	整理 MATLAB 内存碎块
type	显示指定文件的内容

2. 指令中的标点符号

空格　　输入量之间的分隔符。

逗号　，　要显示结果的指令与其后指令之间的分隔；或者输入量之间的分隔；或者数组元素之间的分隔符号。

分号　；　不显示计算结果指令的结尾标志；或者不显示计算结果与其后指令的分隔。或者数组的行间分隔。

冒号　：　生成一维数组；或者表示某种下标。

注释号 %　其后的所有物理行部分都视为注释信息。

单引号 ''　表示字符串。

续行号 …　将其下的物理行看作该行的逻辑继续，当不能在一行完成指令的输入时，采用续行号，在接下来的一行里继续输入指令。

3. 指令行的编辑

由于 MATLAB 是一种交互性的解释性语言，输入指令立刻给出计算结果是它的主要工作方式。为了用户输入方便，MATLAB 设计了如下的功能：

- MATLAB 将所有命令窗口中输入的指令和运行产生的变量存储在专门的工作空间中，以供随时调用。
- 已输入过的指令可以通过键盘上的 ↑ 和 ↓ 键随时调用。
- MATLAB 工作空间中记录的输入指令，同时显示在桌面环境的命令历史窗口中，用户可以随时拷贝到命令窗口中重新执行。

2.3 工 作 空 间

2.3.1 工作空间简介

MATLAB 的工作空间（workspace）是一个十分重要的概念。无论是 M 文件的执行、变量的输入都是在工作空间中进行的。工作空间存在于 MATLAB 的运行环境中，其管理由 MATLAB 本身进行，我们不需要了解其中的细节。这种存在于 MATLAB 整个工作环境中的工作空间又称之基本工作空间，简称工作空间。初学 MATLAB 的用户可以将基本工作空间看成是整个工作环境中的全局变量的集合。向 MATLAB 输入任何数值或变量，以及 M 文件运行的最终结果都驻留在工作空间中。

由于工作空间中的变量在整个工作环境中都有效，我们常常把 SIMULINK 进行仿真时需要动态设置的参数也放在基本工作空间中，修改工作空间中这些参数的值就可以随时了解仿真结果的变化。

可以将工作空间的所有变量以数据文件格式保存起来，在需要时又可以将它装载进工作空间。

MATLAB 6.0 增强了对工作空间的管理，让用户更有效的进行数据的处理。

2.3.2 基本指令

who 和 whos 指令可以用来列出 MATLAB 工作空间中驻留的变量清单。二者的不同在于：whos 在给出驻留的变量名的同时，还给出了各个变量的维数及属性。

例2.8：用who查询MATLAB工作空间中的内存变量

```
>>who
Your variables are:
R          Y          a          student    x          z
X          Z          ans        t          y
```

例2.9：用whos查询MATLAB工作空间中的内存变量，包括它们的维数、大小等属性

```
>>whos
  Name        Size        Bytes  Class

  R           31x31        7688  double array
  X           31x31        7688  double array
  Y           31x31        7688  double array
  Z           31x31        7688  double array
  a           2x3            24  cell array
  ans         1x1           236  struct array
  student     2x3           224  struct array
  t           1x501        4008  double array
  x           1x31          248  double array
  y           1x31          248  double array
  z           1x30          240  double array

Grand total is 4473 elements using 35980 bytes
```

2.3.3 使用工作空间浏览器

MATLAB 5.x有一个专门用于MATLAB工作空间管理的图形用户界面的工具。称为工作空间浏览器（workspace Browser），MATLAB 6.0则将它集成到桌面环境中。

2.3.4 工作空间的保存

用户在使用MATLAB的过程中可以随时将工作空间中的所有变量以数据文件形式保存起来。具体做法是选择桌面环境中的“File：Save workspace As”菜单项。系统将弹出标准的文件选择目录，用户输入适当的文件名后将生成的数据文件（扩展名为.mat）存放在选择的目录下。

如果用户需要使用数据文件中的某些变量，可以将数据文件重新装载进工作空间。方法是选择桌面环境中的“File：Load workspace”菜单项，用户在弹出的对话框中选择需要载入的数据文件。如果这时工作空间中有与待载入的数据文件中同名的变量，原有的变量将会被载入的变量替换

如果用户只想对工作空间中或数据文件中的某些变量进行保存和载入操作，则可以使用save和Load指令，其用法如下：

```
save FileName                   %将全部变量保存为FileName文件
save FileName v1 v2             %把变量v1、v2保存为FileName文件
save FileName v1 v2 –append     %将变量v1、v2添加到FileName文件中
save FileName v1 v2 –ascii      %把变量v1、v2以8位ASCII形式保存为FileName
```

文件

```
save FileName v1 v2 -ascii -double %把变量 v1、v2 以 16 位 ASCⅡ形式保存为 FileName
                                   文件
load FileName                      %将 FileName.mat 文件中所有变量装载进工作空间
load FileName v1 v2                %将 FileName.mat 文件中 v1、v2 装载进工作空间
load FileName v1 v2 -ascii         %将 FileName ASCⅡ文件中 v1、v2 装载进工作空间
```

说明：-ascii 选项可以使数据以 ASCⅡ文本形式存放，这种 ASCⅡ文本文件可以在任何字处理软件中使用和修改。这使得对于尺寸很大的变量的修改很方便。

Save 和 Load 指令具有很强的灵活性，用户甚至可以在 M 文件中包含这些指令，让程序根据需要保存和装载相应的数据文件。

2.4 路 径 设 置

2.4.1 路径设置简介

初学 MATLAB 的人经常遇到这样的问题，好不容易编写了一个 M 文件，保存在相应的目录后（不是 MATLAB 的标准目录），在命令窗口中输入相应的文件名，却发现无法运行，MATLAB 给出的出错信息提示，该指令为没有定义的文件或变量，或者说找不到相应的文件。这里就涉及到 MATLAB 的文件管理和路径设置问题。

MATLAB 的内容十分丰富，当用户在命令窗口中无论是输入一个 M 文件名，还是一个变量，MATLAB 都能够自动识别输入的是指令、M 文件还是变量，并且在庞大的数据库或函数库中找到相应的文件或变量。这里依靠的是自身的文件和搜索路径管理功能。

当用户在命令窗口输入一个指令或变量时，MATLAB 通常的做法是首先查看工作空间，看它是不是已经存在的变量，如果不是，则检查它是否是内建函数，如果都不是，则在 MATLAB 的当前目录和搜索目录中查找是否存在同名的 M 文件。如果这时无法找到相应的文件，MATLAB 将报错说明输入的指令无法识别。

本节开始遇到的问题就很可能是新建的文件不在 MATLAB 的搜索目录中，解决的办法可以将该文件所在目录设置成 MATLAB 的搜索目录，或将该文件移到 MATLAB 能够搜索到的目录下。

2.4.2 目录的设置

MATLAB 具有丰富的文件和层次目录，各种执行程序、工具箱和帮助文档经过分门别类，依次存放在不同的子目录下。我们在使用 MATLAB 进行仿真和计算时，最好创建自己的独立于 MATLAB 的工作目录，如 matlab\mydir，这样不会破坏原有的目录结构，使系统保持清晰的层次结构。

为了使用户自己目录上的 M 文件、数据文件以及 SIMULINK 模型文件与 MATLAB 环境自由地进行交互调用，最简单的方法是将用户目录设置称为当前目录。设置方法是在命令窗

口中输入指令 cd d:\matlab\mydir，或 pwd d:\matlab\mydir。

另一种方法则是将用户目录设置为 MATLAB 的搜索路径中，方法是在命令窗口中输入指令 path(path, 'd:\matlab\mydir')。值得注意的是以上方法设置的目录信息只是在当前 MATLAB 环境中有效，一旦 MATLAB 重新启动，以前的设置就无效，需要重新进行设置。避免这种情况的发生，可以采用路径浏览器进行修改。

2.4.3　路径浏览器的使用

MATLAB 包含的路径浏览器（path brower）可以用来修改和设置 MATLAB 的目录信息。以下方法可以用来启动路径浏览器。

- 在命令窗口中使用 pathtool 指令。
- 在桌面环境中选择"File：Set Path"菜单项。

在路径浏览器中我们可以修改当前的工作目录，也可以增加或删除搜索路径中的目录项。当然，当前目录的修改是临时的，而进行搜索路径的修改后，MATLAB 将提醒用户是否要保存当前的修改，如果用户确定后，对于搜索路径的修改就可以保存下来。下次 MATLAB 启动时，不需要重新进行修改。

例如，如果用户要在搜索路径中包含自己的工作目录，可以按下"Add Folder…"按钮直接在系统目录中寻找并选择自己的工作目录，也可以在如图 2.5 所示的添加目录框中直接输入自己的工作目录。

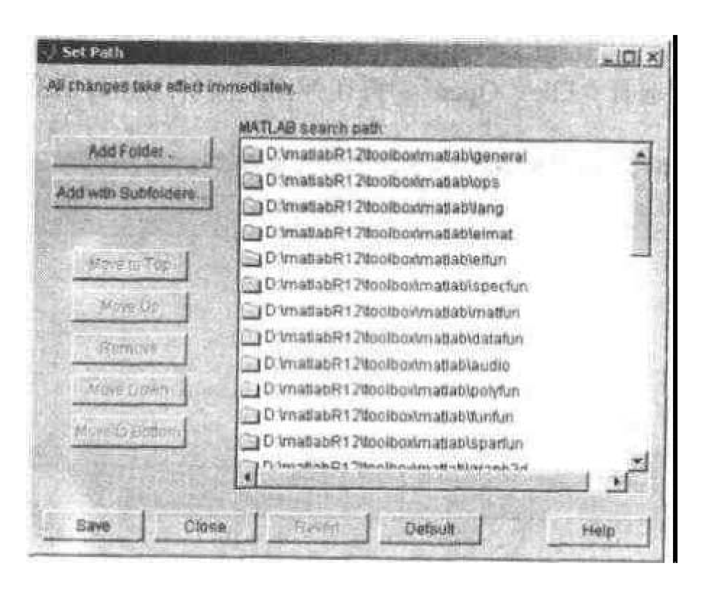


图 2.5　MATLAB 的搜索路径设置对话框

2.5　M 文件的编写与调试

2.5.1　M 文件编辑器

从 2.2 节我们知道，对于比较简单的问题，可以通过在命令窗口直接输入一组指令去求

解。但是当要解决的问题所需的指令很多，并且存在诸如循环、判断等比较复杂的结构时，或者当一组指令通过改变少量参数即可解决不同问题时，直接在指令窗口中输入这些指令就不大方便了。为此，需要编写 M 文件。

所谓 M 文件就是以文本形式记录 MATLAB 指令的文件，后缀名是“.m”。通常我们所说的 MATLAB 程序就是指这种 M 文件。我们只要在命令窗口中输入该文件名（该文件在 MATLAB 的搜索路径中），MATLAB 就可对文件中的命令依次执行。与在指令窗口直接运行指令一样，M 文件运行产生的变量也驻留在工作空间中。

MATLAB Editor/Debugger 是一个集编辑与调试功能于一体的工具环境，如图 2.6 所示。它不仅可以完成基本的文本编辑操作，还可以对 M 文件进行调试。从 MATLAB 5.3 开始，Editor/Debugger 就可以在用户编写过程中自动识别其中的语法结构，并用不同的颜色表示出来，称之为 Syntax highlighting。如注释（以“%”符号开头）用绿色表示，关键字用蓝色表示等等。

为创建新的 M 文件，启动编译器的方法有：

- 在命令窗口输入指令 edit。
- 单击桌面工具条上的新建文件图标。
- 在桌面菜单中选择“File：New”，再选择“M-File”。

打开已有的 M 文件的方法有：

- 在命令窗口中输入指令 edit filename，filename 是待编辑的文件，可以不带文件扩展名。
- 单击桌面工具条上的打开文件图标，在弹出的文件选择对话框中选择待编辑的文件。
- 在桌面菜单中选择“File：Open”，再在弹出的文件选择对话框中选择待编辑的文件。

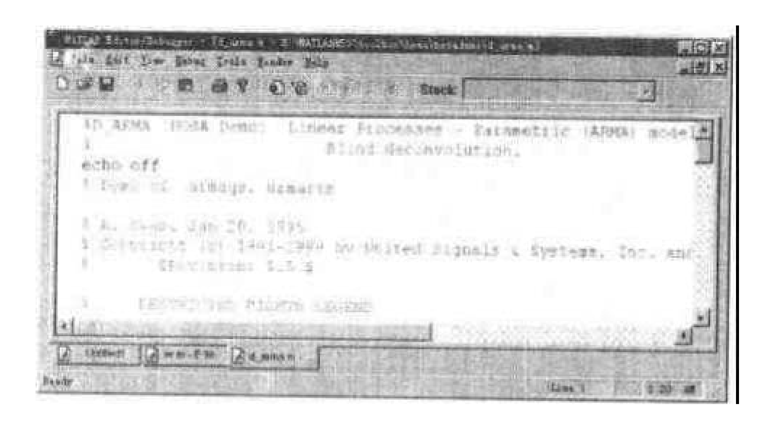


图 2.6　M 文件编辑/调试器

2.5.2　MATLAB 控制流结构

同其他计算机语言一种，MATLAB 的 M 文件允许程序员根据某些判断来控制程序流的执行次序。MATLAB 提供了几种控制结构，依次为：

（1）循环结构：

for x=1：4

…

```
end
while expression
  …
end
```

（2）选择结构:

```
if   expression
…
else
…
end
```

（3）条件判断结构

```
switch x                %x 为需要判断的变量
  case result1          %当 x 等于 result1 时，执行下面的指令，然后跳出该结构。
    …
  case result2
    …
…
  case resultn
    …
  otherwise             %当 x 不满足以上所有条件时，缺省执行的指令块。
    …
end
```

> 说明：MATLAB 的 switch-case 结构与 C 语言中不同，它不需要 break 指令就可跳出该结构。

（4）错误控制结构

```
try
  …
catch
  …                     %当 try 中的指令块出现错误时，该指令块被执行，可以用 lasterr
                        指令查询发生错误的原因。
end
```

其他常用控制指令：

- input

此指令将 MATLAB 的控制权暂交给用户。此后，用户通过键盘输入相应的数值或表达式，并回车输入工作空间，同时将控制权交还给 MATLAB。

- keyboard

该指令将控制权交给键盘，用户可以从键盘输入各种合法的 MATLAB 指令。使用 return 可以将控制权交还给 MATLAB。

- pause 或 pause(n)

该指令暂时停止文件的执行，用户按任意键，文件继续执行，后面的参数 n 表示暂停多少秒。

- break 与 continue

break 该指令将使包含有该指令的 for、while 循环结构的循环过程立即中止。

continue 该指令将使包含有该指令的 for、while 循环结构的本次循环过程结束，立刻进行下一次的循环过程。与 break 不同，continue 不中止整个循环过程。

2.5.3　M 函数文件

采用 M 文件还可以进行自定义函数的编制。与一般 M 文件不同，函数文件接受外界传来的参数，经过处理后将返回值传回调用文件，而内部的计算是不可见的。函数内部的变量在函数执行完后即删除，不会驻留在工作空间中（以 global 关键字声明的全局变量除外）。

函数文件第一行总是以 function 引导的函数头，其中包括函数名、输入参数等，也可以没有任何输入参数。

函数文件执行时，MATLAB 将开辟专门的函数工作空间，所有中间变量均放在函数工作空间中。当函数执行完，或遇到 return 指令时，就退出函数文件的执行，并释放函数工作空间，其中的所用变量均消除。可见，函数文件的执行是相对独立的，虽然不同函数文件中的变量名可能相同，但它们是在各自工作空间中运行的不同变量。

函数文件的保存应该取与函数名相同的文件名。

2.5.4　变量的作用范围

MATLAB 中，根据变量的作用范围可将变量分为局部变量和全局变量。

（1）局部（Local）变量

存在于函数空间内部的中间变量，产生于该函数的运行过程中，其作用范围仅仅限于该函数本身。没有经过特殊声明的变量，缺省状态都是局部变量。

（2）全局（Global）变量

通过 global 指令，MATLAB 允许不同的函数空间以及基本工作空间共享一个变量。这种变量称为全局变量。每个希望共享全局变量的函数或 MATLAB 基本工作空间，必须逐个用 global 指令进行声明。否则，没有声明的函数或 M 文件将不能使用该全局变量。

如果某个函数使全局变量发生变化，则其他所有函数空间以及基本工作空间的同名全局变量也随之变化。

除非与全局变量联系的所有工作空间都被删除，否则全局变量依然存在。

对全局变量的声明必须在该变量使用前进行声明，一般放在函数文件或 M 文件的开头，并且采用大写与一般变量区分开。

全局变量虽然给函数之间的参数传递带来了方便，但破坏了函数本身的封装性，不符合面向对象的编程思想，因此，除非万不得已，一般不采用全局变量。

2.5.5　M 文件的调试

用户在编写 M 文件时，经常会遇到这样那样的错误。有的错误即语法错误，如指令语法使用错误、符号的漏写等等，MATLAB 在运行时能够立即发现，并指出错误发生的位置和原因。但是有的错误是算法本身的一些错误，如我们对 MATLAB 指令的使用错误、对算法的理解错误以及算法设计的一些问题，其表现形式也是多样的，有的程序正常运行，但结果不

对，有的导致程序不能正常运行。尤其是函数文件执行后，其中间变量全部被删除，我们无法通过其中变量的变化来查找函数中的问题。这时对M文件的调试就显得很重要。

MATLAB从4.0起就提供了专门的指令式调试工具，从5.x开始使用更为简单的图形用户界面（GUI）完成对程序的调试。MATLAB 6.0则采用了更为友好的界面，使用户的调试过程更加方便。

1. 断点设置

把光标放在需要设置断点的地方，单击工具条上的断点设置图标，便可在该行的最左端出现红色断点标志，如图2.7所示。

2. 开始调试

在命令窗口中执行需要调试的文件，该文件将在遇到的第一个断点处停下来，如图2.8所示。之后程序进入调试阶段，用户可以单步执行程序，也可直接跳到下一个断点处。在执行过程中，用户可以通过观察变量值的变化查找文件中的错误。

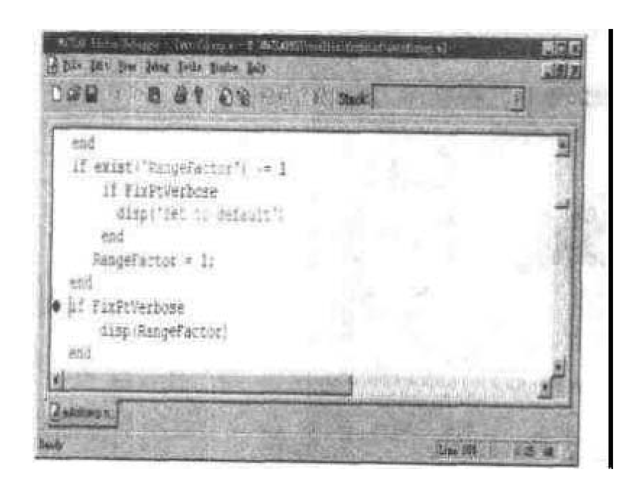


图2.7　在M文件中设置断点

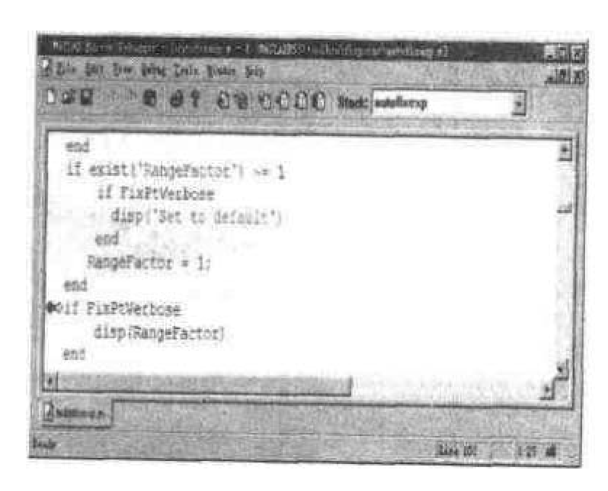


图2.8　程序执行到断点处

3. 运行中中间变量的观察

在调试窗口菜单“Tool：Option：General：Show DataTips”被选中的情况下，将鼠标移到相应的待观察变量上面，调试器会自动弹出该变量的结果，如图2.9所示。

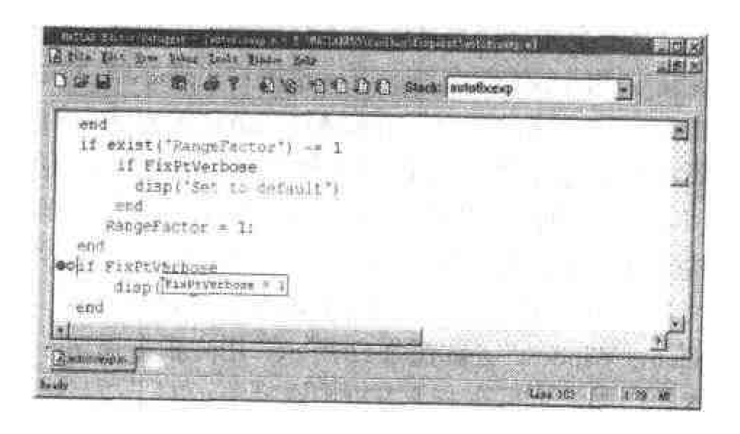


图2.9　中间变量的观察

当然，也可以在程序中断运行情况下，在命令窗口中输入待观察的变量名，同样可以得到该变量的中间值，不过这种方法不及上面的方法方便。

2.6 在线演示和帮助

2.6.1 在线引导

对于MATLAB的初级用户，学习MATLAB基本使用方法的最简单方法，就是在命令窗口中输入intro指令，系统将会产生一个图形用户界面的在线引导如图2.10所示。通过单击右边的按钮，用户可以逐页进行各种常用指令的学习。每页的上面显示模拟命令窗口中的情况，包括输入的指令和计算结果，下面则是相应的文字解释。通过在线引导，用户可以很快熟悉MATLAB一些基本指令的用法，为今后更深入的学习打下基础。

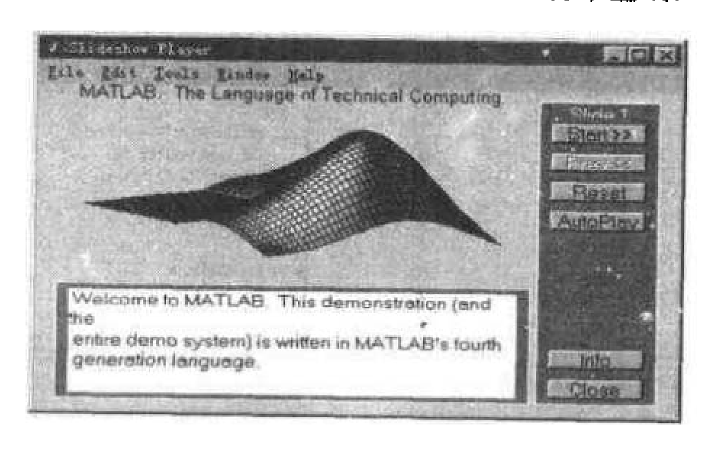


图2.10　MATLAB在线引导对话框

2.6.2 演示程序

Mathworks公司为了能使用户尽可能快的掌握MATLAB系列产品的使用，在不同的版本推出的同时也设计了一组旨在介绍MATLAB诸多功能的演示程序。运行这些程序，观察程序运行的结果，然后研究实现演示的源文件，可以让用户迅速了解MATLAB的编程技巧，使编写的程序更加规范、高效。这也是迅速提高MATLAB编程能力的一条捷径。

为了实现对这些演示程序的管理，Mathworks公司推出了专门的图形用户界面MATLAB Demos来运行所有的演示程序，而把相应的源文件放在各自工具箱或相关目录下。系统在第一次启动时会自动搜索所有目录下的MATLAB演示程序，并将它们组织到MATLAB Demos中。在命令窗口中输入“demo”，或单击“Help: Examples and Demos”菜单项，就可以打开演示窗口“MATLAB Demo Window”，如图2.11所示。用户根据需要，可以在演示窗口的左边方框中选择演示内容的门类，然后在右下方框中选择演示的子类，最后再单击右下方的运行按钮，即可启动相应演示程序。

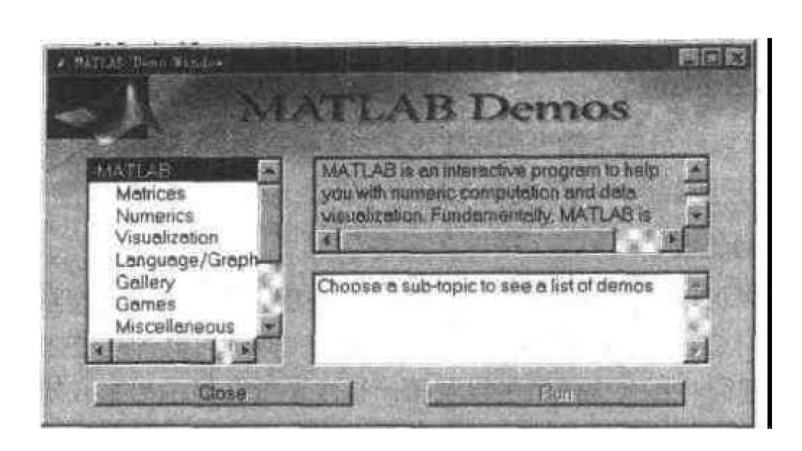


图2.11 MATLAB中的演示程序

2.6.3 帮助系统

MATLAB 从一开始就为用户提供了各种详细充实的帮助，用户可以很容易地获取MATLAB 各种产品的使用方法以及各种指令的有关信息。一般有3种不同方法获得帮助。

（1）在命令窗口中直接运行 help 指令，获取有关指令的使用方法。基本的操作是：help command，command 是需要了解的指令。也可以使用 lookfor 指令。

（2）运行帮助浏览器（Help brower）。

（3）查看 PDF 帮助文档。

1. 直接获取帮助

如果用户知道所要查询的指令名称，那么使用 help 是获得在线帮助的最简单有效的途径。例如，下面的例子显示了对二维图形绘制指令 plot 的帮助信息。

```
>>help plot
PLOT    Linear plot.
    PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.  If X is a scalar and Y is a vector, length(Y)
    disconnected points are plotted.
...
          y     yellow        .     point          -     solid
          m     magenta       o     circle         :     dotted
          c     cyan          x     x-mark         -.    dashdot
...
See also SEMILOGX, SEMILOGY, LOGLOG, GRID, CLF, CLC, TITLE,
XLABEL, YLABEL, AXIS, AXES, HOLD, COLORDEF, LEGEND, and SUBPLOT.
```

显示的在线帮助内容中，指令名称采用大写字母（如 PLOT）给出，这主要是为了使指令名称比较容易识别。而在 MATLAB 命令窗口中输入的所有函数和指令都必须用小写字母。

在每个指令在线帮助的最后部分，一般会列出与该指令相关的其他指令名称。

用help指令的得到的在线帮助系统实际上是定义该指令的M文件中第一段以%开始的注

释部分，这里一般放置的是该指令的描述信息与使用方法。读者可以通过 M 文件编辑器打开 plot.m 文件来看一看。

help 指令除了可以获得单个指令的在线说明，还可以运行不带任何限定的 help 指令，来获得指令分类名称的列表。

2. lookfor 指令的使用

lookfor 指令可以可以根据用户提供的完整或不完整的关键词，去搜索与之相关的指令。其搜索原理是：对 MATLAB 目录中的每一个 M 文件注释区的第一行进行扫描，一旦发现这行中包含指定的关键词，那么该函数名以及第一行注释就会显示出来。如果使用-all 参数，则对整个注释段进行搜索。这样会搜索到更多的相关指令，不过搜索的时间也相应增加。例如，下面是对积分的相关指令的搜索情况

```
>>help integral
ELLIPKE Complete elliptic integral.
EXPINT Exponential integral function.
DBLQUAD Numerically evaluate double integral.
INNERLP Used with DBLQUAD to evaluate inner loop of integral.
QUAD     Numerically evaluate integral, low order method.
QUAD8    Numerically evaluate integral, higher order method.
COSINT Cosine integral function.
SININT Sine integral function.
ASSEMA Assembles area integral contributions in a PDE problem.
COSINT Cosine integral function.
FOURIER Fourier integral transform.
IFOURIER Inverse Fourier integral transform.
SININT Sine integral function.
BLKPIDCON The output of the block is the sum of proportional, integral and
```

3. 帮助浏览器的使用

MATLAB 6.0 为用户提供了专门的帮助浏览器，用户在使用 MATLAB 的过程中，随时可以如果按下【*F1*】键来获得当前操作或指令的相关帮助信息，如图 2.12 所示。

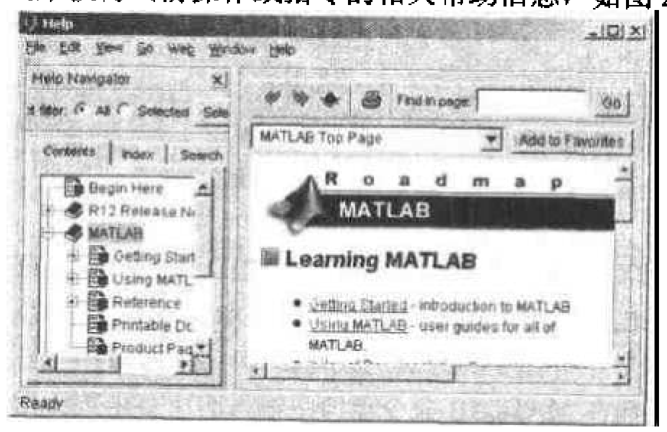


图 2.12　MATLAB 6.0 的帮助浏览器

4. PDF 帮助文档

Mathworks 公司将所有有关 MATLAB 的用户指南和指令手册采用 PDF 文件提供给用户。PDF 文档全部放在\matlab\help\pdf_doc 目录下。阅读 PDF 文档需要 Adobe Acrobat Reader 软件的支持。

假设用户计算机中已经装有 Acrobat Reader 软件，则可以通过以下方式打开 PDF 文档：

在"资源管理器"中打开 PDF 所在目录\matlab\help\pdf_doc，选择需要查看的 PDF 文档，例如对于 SIMULINK 需打开的文件为 SIMULINK.pdf，用鼠标双击文件的图标。

也可以先运行 Acrobat Reader 条件，然后在 Acrobat Reader 环境中找到\matlab\help\pdf_doc 目录下所需的文件。

一旦打开所需的 PDF 文档，用户可以在其中浏览相应的帮助信息，也可以通过打印进行离线阅读。

MATLAB 的 PDF 文档所提供的内容十分详尽，涉及到 MATLAB 的各个方面，因此每一个想要深入学习 MATLAB 使用的用户，需要仔细研究一下有关的 PDF 文档。

图 2.13 显示的是打开 simulink.pdf 文档的情况。

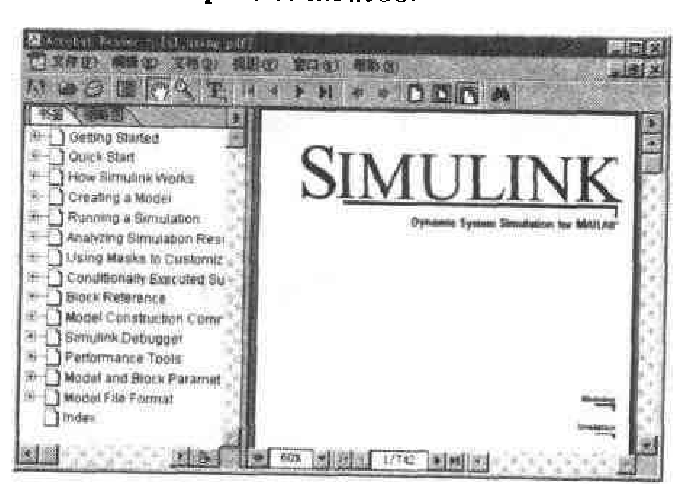


图 2.13　用 Acrobat Reader 4.0 打开 SIMULINK.PDF 文档

第3章　SIMULINK 4.0 概述

SIMULINK 是 MATLAB 系列工具软件包中最重要的组成部分。它能够对包括连续系统、离散系统以及连续离散的混合系统进行充分的建模与仿真；能够借助其他工具直接从模型中生成可以直接投入运行的执行代码；可以仿真离散事件系统的动态行为；在众多专业工具箱的帮助下完成诸如 DSP、电力系统等专业系统的设计与仿真。可以说，SIMULINK 自诞生起就完全建立在 MATLAB 的基础上，并随着 MATLAB 的不断更新而发展，功能越来越强大，使用也越来越方便。

随同 MATLAB 6.0 一道发行的是 SIMULINK 4.0 版本。该版本在 3.0 版本的基础上，保持了大的框架不变，而增加了某些模块对矩阵信号的支持，同时改进了用户界面。因此，熟悉 3.0 版本的用户可以很快过渡到 4.0 版本中。

由于本书是面向初级用户编写的，因此在具体介绍 SIMULINK 之前，首先介绍动态系统交互式仿真工具 SIMULINK 4.0 的工作环境、基本组成、常用工具和使用方法等。使初学的读者对该软件包有一个初步的认识。

3.1　SIMULINK 4.0 导引

3.1.1　SIMULINK 4.0 的安装

SIMULINK 的安装是同 MATLAB 的安装过程同时进行的。启动 MATLAB 的安装过程后，将出现如图 1.2 所示的安装先导。在有关 SIMULINK 的选项前打勾，则表示选中了安装 SIMULINK 项，安装向导在安装 MATLAB 的同时会自动安装 SIMULINK 软件包。为了能够随时获取相关的帮助文件，建议选择安装 SIMULINK 的帮助文档。

3.1.2　SIMULINK 4.0 的启动

在 MATLAB 环境中启动 SIMULINK 的方法是：

- 在 MATLAB 6.0 的桌面环境中单击工具按钮。
- 在命令窗口中输入 simulink 指令。

SIMULINK 启动后首先出现的是 SIMULINK 库浏览器（SIMULINK Library Browser），如图 3.1 所示。

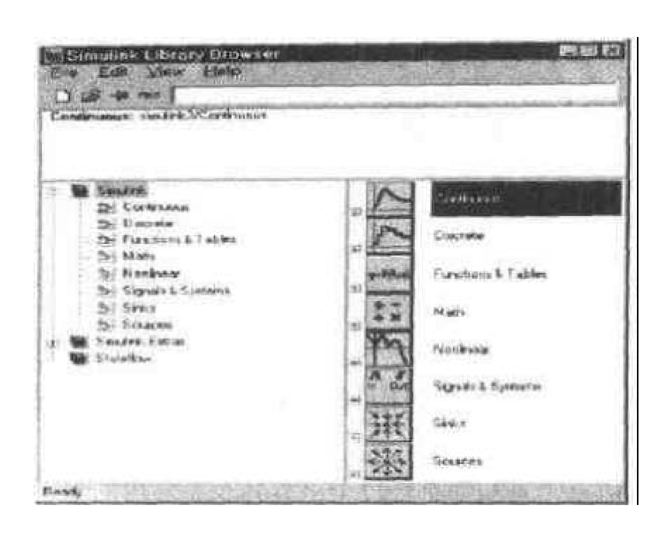


图 3.1　SIMULINK 中的库浏览器

3.1.3　SIMULINK 4.0 的工作环境

SIMULINK 的工作环境主要是由库浏览器与模型窗口组成。前者为用户提供了展示SIMULINK 标准模块库和专业工具箱的界面，而后者是用户创建模型方框图的主要地方。

1. SIMULINK 的库浏览器

2. SIMULINK 的模型窗口

在库浏览器中单击工具条图标 D，即可创建一个新的模型窗口（如图 3.2 所示）。可以在该窗口中完成模型方框图的绘制。

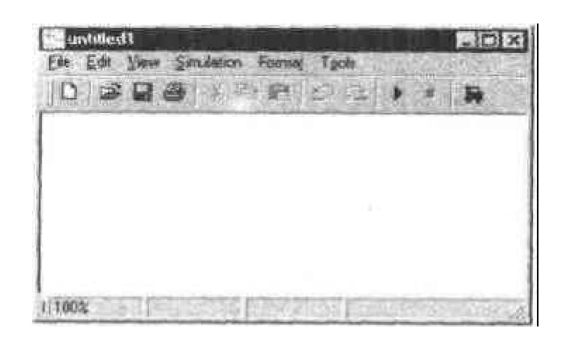


图 3.2　新的模型窗口

3.1.4　SIMULINK 4.0 的演示程序

学习 SIMULINK 的最好方法是仔细研究和阅读随带的演示程序。SIMULINK 为了方便读者在尽可能短的时间了解它的强大功能，提供了大量的演示模型，这些模型几乎涵盖了SIMULINK 的方方面面，读者在其中可以学到各种系统，尤其是复杂系统建模的思想和技巧。读者在学习如何使用 SIMULINK 进行建模与仿真之前，可以打开 SIMULINK 4.0 的演示模型，看看 SIMULINK 的仿真框图，同时试着运行仿真，通过观察仿真结果来体会 SIMULINK 的不同于一般仿真软件的独特之处。

3.2　SIMULINK 4.0 的组成

3.2.1　应用工具箱

SIMULINK 软件包的一个重要特点是它完全建立在 MATLAB 的基础上，因此，MATLAB 各种丰富的应用工具箱也可以完全应用到 SIMULINK 环境中来，这无疑大大扩展了 SIMULINK 的建模和分析能力。

基于 MATLAB 的所有工具箱都是经过全世界各个领域内的专家和学者共同研究的最新成果，每一个工具箱都可谓是千锤百炼，其领域涵盖了自动控制、信号处理和系统辨识等十多个学科，并且随着科学技术的发展，MATLAB 的应用工具箱始终处在不断发展完善之中。

MATLAB 应用工具箱的另一个特点是完全的开放性，任何用户都可随意浏览、修改相关的 M 文件，创建满足用户特殊要求的工具箱。由于其中的算法有很多是相当成熟的产品，用户可以采用 MATLAB 自带的编译器将其编译成可执行代码，并嵌入到硬件当中直接进行执行。

3.2.2　实时工作室

SIMULINK 软件包中的实时工作室（real-time workshop）可以将 SIMULINK 的仿真框图直接转化成 C 语言代码，从而直接从系统仿真过渡到系统实现。该工具支持连续、离散和连续-离散混合系统。用户完成 C 语言代码的转换后，可以直接进行汇编，生成执行文件。

借助实时工作室，用户可以：

- 无需手工编写代码和复杂的调试过程，就可以完成从动态系统设计到最后代码实现的全过程，包括控制算法设计、信号处理研究动态系统都可以借助 SIMULINK 的可视化方框图进行方便的设计。
- 一旦 SIMULINK 完成了系统的设计，用户可以采用借助工具生成嵌入式的代码，在进行编译、连接之后，直接嵌入到硬件设备当中。
- 异步仿真。由于 MATLAB 中可以以 ASCⅡ或二进制文件记录仿真所经历的时间，用户可以将仿真过程放在客户机或发送到远程计算机中进行仿真。

SIMULINK 4.0 的实时工作室支持大量的不同的系统和硬件设备，并且具有友好的图形用户界面，使用起来更加方便、灵活。

3.2.3　状态流模块

与 MATLAB 6.0 使用的 Stateflow 为 4.0 版本。SIMULINK 的模块库中包含了 Stateflow 的模块，用户可以在该模块中设计基于状态变化的离散事件系统。将该模块放入 SIMULINK 模型当中，就可以创建包含离散事件子系统的更为复杂的模型。

3.2.4 扩展的模块集

如同众多的应用工具箱扩展了 MATLAB 的应用范围，Mathworks 公司为 SIMULINK 提供了各种专门的模块集（BlockSet）来扩展 SIMULINK 的建模和仿真能力。这些模块集涉及到电力、非线性控制、DSP 系统等不同领域，满足了 SIMULINK 对不同系统仿真的需要。

这些模块集包括：

- CDMA Reference Blockset

用于满足 IS-95A 无线通信标准系统的设计与仿真的模块库。

- Communications Blockset

满足对通信系统物理层建模与仿真需要的模块库。

- Dials & Gauges Blockset

用于监控 SIMULINK 信号与模块参数的图形装置。

- DSP Blockset

用于数字信号处理系统仿真和设计的模块库。

- Fixed-Point Blockset

用于固定点系统仿真、设计和代码生成的模块库。

- Nonlinear Control Design (NCD) Blockset

为非线性系统的设计提供时域优化方法，它可以基于指定的时域性能约束自动调整系统参数。

- Power System Blockset

用于电力系统设计和仿真的模块库。

3.2.5 SB2SL 工具

SB2SL 作为 SIMULINK 仿真软件包的一部分，可以将 SystemBuild 的 SuperBlocks 模块转化成 SIMULINK 仿真模块。在 MATLAB 6.0 中，SB2SL 2.1 为 Real-Time Workshop 的使用提供了更完善的支持。如果用户使用 SB2SL 工具将 SystemBuild 模型转化成 SIMULINK 模型，并且使用 Real-Time Workshop 工具创建代码，则这些代码绝大多数是从转化后的模型创建的。

3.3 SIMULINK 中的基本概念

3.3.1 模块与模块框图

SIMULINK 模块框图是动态系统的图形显示，由一组称为模块的图标组成，模块之间采用连线联结。每个模块代表了动态系统的某个单元，并且产生一定的输出。模块之间的连线表明模块的输入端口与输出端口之间的信号连结。模块的类型决定了模块输出与输入、状态和时间之间的关系。一个模块框图可以根据需要包含任意类型的模块。

模块代表了动态系统某个功能单元，每个模块一般包括一组输入、状态和一组输出等几

个部分，如图 3.3 所示。模块的输出是仿真时间、输入或状态的函数。



图 3.3　模块的基本结构

模块中的状态是一组能够决定模块输出的变量，一般当前状态的值决定于以前时刻的状态值或输入，这样，具有状态变量的模块就必须存贮以前时刻的输入或状态值，这样的模块我们称为记忆功能模块。

例如 SIMULINK 中的积分（Integrator）模块就是典型的记忆功能模块。模块当前的输出是该模块从仿真开始到当前时刻这一时间段内输入信号的积分。当前时刻的积分取决于历史输入，因此积分就是该模块的一组状态变量。另一个典型的例子就是 SIMULINK 中的单纯记忆（Memory）模块，该模块能够存储当前时刻的输入值，并在将来的某个时刻进行输出。该模块中的状态变量就是输入的历史值。

SIMULINK 当中的增益（Gain）模块是无状态变量的典型例子。增益模块的输出完全由当前的输入值决定，因此不存在状态变量。其他的无状态变量的模块还有：求和模块（Sum）和点乘模块（Product）等。

SIMULINK模块的基本特点是参数化的，许多模块都具有独立的属性对话框，在对话框中用户可定义模块的各种参数，例如，增益模块中的增益参数，这种调整甚至可以在仿真过程中实时进行，从而让用户能够找到最合适的参数值。这种能够在仿真运行过程中实时改变的参数又被称为可调参数（tunable parameter），可以由用户在模块参数中任意指定。

SIMULINK还允许用户创建自己的模块，这个过程又称为模块的定制。定制模块不同于SIMULINK中的标准模块，它可以由子系统封装得到，也可以采用M文件或C语言实现自己的功能算法，称之为S函数。用户可以为定制模块设计属性对话框，并将定制模块合并到SIMULINK库中，使得定制模块的使用与标准模块的使用完全一样。

3.3.2　信号

SIMULINK使用“信号”一词来表示模块的输出值。SIMULINK允许用户定义信号的数据类型（8位，16位或是32位），数值类型（实数还是复数）和维数（1维数组还是2维数组）等等。需要注意的是，有的模块只能接受特定类型的信号。

数据类型（data type）是计算机中数据的内部表示形式，SIMULINK 可以处理任何 MATLAB 支持的内建（built-in）数据类型的参数值和信号值，例如 8 位整型、双精度和布尔量。SIMULINK 定义了两个指定的数据类型，它们能够存储特定的信息，即：

- Simulink.Parameter
- Simulink.Signal

SIMULINK 允许用户创建 Simulink 数据类型的实例（称为数据对象）来作为模块中的参数和信号变量。

3.3.3 求解器

SIMULINK 模块指定了连续状态变量的时间导数，但本身没有定义这些导数的具体值，它们必须在仿真过程中通过微分方程的数值求解方法计算得到。SIMULINK 提供了一套高效、稳定、精确的微分方程数值求解方法（ODE），用户可以根据需要和模型特点选择最适合的求解算法。

3.3.4 子系统

SIMULINK 允许用户在子系统的基础上构造更为复杂的模型。其中每一个子系统都是相对完整的，完成一定功能的模块框图。通过对子系统的封装，用户还可以实现带触发或使能功能的特殊子系统。子系统的概念体现了分层建模的思想，是 SIMULINK 的重要特征之一。

3.3.5 零点穿越

在 SIMULINK 对动态系统进行仿真的过程中，一般在每一步仿真中都会检查系统状态变化的连续性。如果 SIMULINK 检测到了某个变量的不连续性，为了保持状态突变处系统仿真的准确性，仿真程序就会自动调整仿真步长，适应这种变化。

动态系统中状态的突变对系统的动态特性具有重要的影响，例如弹性球在撞击地面时其速度方向会发生突变（朝向相反方向），这时，如果撞击的时刻不是正好发生在仿真时刻当中（在相邻两步仿真之间），SIMULINK 的求解算法就不能正确反映系统的特性。如果采用固定步长的算法，求解器就不能对此作相应的处理；相反，如果我们使用变步长的求解算法，SIMULINK 就会在确定突变时刻之后，在突变前后增加额外的仿真计算，以保证突变前后计算的准确性，变步长的求解算法在状态变化缓慢时会增加仿真的步长，而在状态变化剧烈时减小仿真的步长，这样，在系统突变时刻，过小的仿真步长将会导致仿真时间的增加。

SIMULINK 采用一种称为零点穿越检测（zero crossing detection）的方法来解决这个问题。

采用这种方法，模块首先记录下零点穿越的变量，每一个变量都是有可能发生突变的状态变量的函数。在突变发生时，零点穿越函数也从正数或复数穿过零点。通过观察零点穿越变量的符号变化，就可以判断仿真过程中系统状态是否发生突变的现象发生。

如果检测到零点穿越事件发生，SIMULINK 将通过对变量的以前时刻和当前时刻的插值来确定突变发生的具体时刻。然后，SIMULINK 调整仿真的步长，逐步逼近并跳过状态的不连续点，这样就避免了直接在不连续点上进行仿真。因为，对于不连续系统而言，不连续点处的状态值可能是没有定义的。采用零点穿越检测技术，SIMULINK 可以准确地对不连续系统进行仿真。许多模块都支持这种技术，从而很大提高了系统仿真的速度和精度。

零点穿越一般具有方向属性。包括：

- 上升穿越：信号沿上升方向穿越零点，或信号由零变成正数。
- 下降穿越：信号沿下降方向穿越零点，或信号由零变成负数。
- 任意穿越：上升穿越或下降穿越，满足二者之一的条件。

> 注意：过大的容许误差可能会使 Simulink 错过某些不连续点的检测。因此，在存在不连续状态的模型当中，用户一般不要将容许误差设置过大。

以下是可能发生零点穿越的模块：

Abs（绝对值模块）
Backlash（回滞模块）
Dead Zone（死区模块）
Hit Crossing（碰撞穿越模块）
Integrator（积分模块）
MinMax（取最大最小值模块）
Relay（延时模块）
Relational Operator（关系运算符模块）
Saturation（饱和模块）
Sign（取符号模块）
Step（阶跃信号模块）
Subsystem（子系统模块）
Switch（切换开关模块）

3.4　SIMULINK的常用工具

3.4.1　仿真加速器

SIMULINK 加速器能够提高模型仿真的速度，它的基本工作原理是利用 Real-Time Workshop 工具将模型方框图转换成 C 语言代码，然后采用编译器将 C 代码编译成可执行代码，由于用可执行代码取代了原有的 MATLAB 解释器，仿真的速度可能会有本质的提高。

SIMULINK 既可以工作在正常模式（Normal），也可以工作在加速模式（Accelerator）下。工作在加速模式下时，SIMULINK 将 C 代码编译成 mex 文件，在编译过程中，对原有的 C 代码进行优化重组，可以极大地提高模型仿真的速度。并且模型越复杂，提高的程度越明显，一般来说，可以将仿真的速度提高 2~6 倍。

为了使 SIMULINK 工作在加速模式下，在模型窗口中选择“Simulation：Accelerator”菜单项。如果要恢复到正常模式，可以选择选择“Simulation：Normal”菜单项。

图 3.4 显示的是 SIMULINK 中的 f14 演示模型。

用户在加速模式下启动仿真后，加速器会自动生成C代码并且进行编译，然后将生成的代码放置在 modelname_accel_rtw 的子目录下（modelname 为具体的模型名称），编译后的 MEX 文件则放在 MATLAB 当前的工作目录下，最后运行编译后的模型。

注意：如果代码编译不成功，最有可能的原因是还没有正确设置 mex 命令。解决的方法是在命令窗口输入 mex –setup 指令，并在列表中选择 C 编译器。

值得注意的是，加速器产生的 C 代码只能作为加速仿真之用，如果需要生成其他作用的 C 代码（例如产生嵌入式 C 代码），就必须使用 Real-Time Workshop 工具。

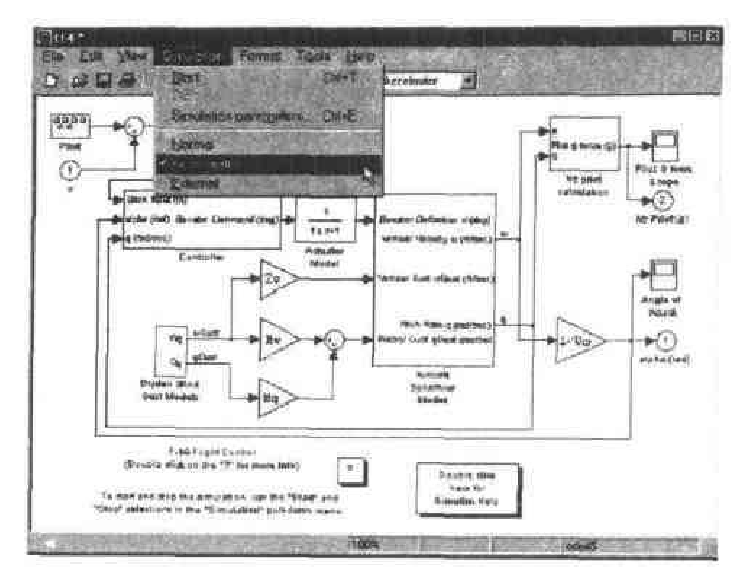


图 3.4 将 SIMULINK 切换到加速模式

如果用户改变了模型的结构，例如增加或删除模型中的模块，加速器会自动修改对应的 C 代码并重新进行编译。下面的操作将会影响模型的结构：

- 改变积分的算法。
- 增减模块或改变模块之间的连结。
- 改变模块输入输出端口的数目。
- 改变模型当中状态变量的数目
- 改变 Trigonometric Function 模块中的函数。
- 改变 Sum 模块中所用的符号。
- 加入 Target Language Compiler™（TLC）文件来产生某个内建的 S 函数。

在模型仿真过程中，如果用户对模型所作的修改影响到模型的结构，SIMULINK 将忽略这种改变，并且出现警告信息。用户如果要想使修改生效，必须停止仿真过程，待修改完成后，重新开始。当然，简单的修改，如改变 Gain 模块的增益值，不会产生警告消息，这表示加速器认可这种修改。

需要注意的是，加速器不会显示仿真过程本身产生的警告消息，如被零除和数据溢出，这点与前面所讲的情况不大一样。

（1）调试过程中使用加速器

如果用户调试的模型很大，采用加速器可以加快调试的进程，例如如果仿真执行到某个断点处的时间很长，就可以采用加速器来尽快达到所要调试的断点处。在调试状态下运行加速器的方法是在“Simulatin”菜单中选择“Accelerator”项，然后在命令窗口中输入：

```
>>sldebug modelname
```

下面的指令将在指定时刻设置断点：

```
>>tbreak 10000
>>continue
```

一旦仿真在断点出挂起后，用户可以使用 emode 指令在正常模式和加速模式之间进行切换。注意，如果是加速模式，则不允许采用单步跟踪的调试步骤。

（2）两种仿真模式的比较

通过 tic, toc 指令，用户可以观察仿真在正常模式与在加速模式下，模型仿真速度的差别，例如，我们以 f14 模型为例，在正常模式下：

```
>>tic,[t,x,y]=sim('f14',1000);toc
elapsed_time =
                14.1080
```

而在加速模式下：

```
elapsed_time =
                6.5880
```

可见，加速模式下的仿真速度是正常模式下仿真速度的两倍以上。以上结果是在 233 MHz Pentium 处理器下得到的。

3.4.2　模型比较工具

SIMULINK 当中的模型比较工具（Model Differencing Tool）可以让用户迅速找到两个模型之间的不同点，例如两个相同模型之间的不同版本。

启动模型比较工具的方法是单击“Tool：Model differences”菜单，出现如图3.5所示的图形界面。

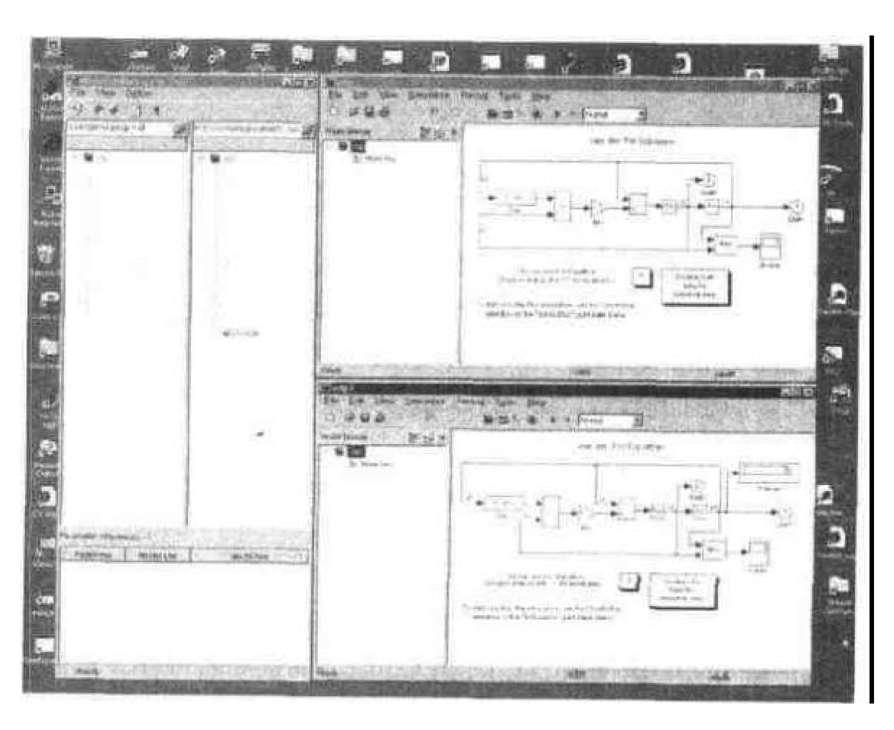


图 3.5　SIMULINK 中的模型比较工具

模型比较工具的界面分成三个子窗口：左上方的子窗口显示的是第一个模型的具体内容；右上方的子窗口显示的是第二个模型的具体内容。下方的子窗口显示的是两个模型当中相同模块的不同参数。SIMULINK采用不同的颜色来表示两个模型中的不同之处：

- 蓝色表示只出现在其中一个模型中的模块。
- 红色表示两个模型当中都具有的模块，但模块的参数不同。
- 绿色表示两个模型当中完全相同的模块。

用户选择“View：HTML Report”菜单，还可以生成对比较结果进行统计的HTML报表。

3.4.3 仿真统计表

SIMULINK 中的仿真统计表生成器（simulation profiler）可以根据仿真过程中的数据生成一个称为仿真统计表（simulation profile）的报告。该报告可以显示该模型的仿真过程中每个功能模块所花费的时间，从而可以让用户确定决定模型仿真速度的主要因素，为进一步优化仿真模型提供帮助。

为了使用该工具，用户首先打开指定的模型，选择“SIMULINK：Profiler”菜单。然后启动仿真。当仿真结束后，SIMULINK 将生成并在 MATLAB 帮助浏览器中显示仿真的统计表。最后，SIMULINK 会将该统计表保存到 MATLAB 的工作目录下。

3.5 SIMULINK 环境的设置

3.5.1 MATLAB 环境设置对话框

MATLAB 环境对话框可以让用户集中设置 MATLAB 及其工具软件包的使用环境，包括 SIMULINK 环境的设置。要在 SIMULINK 环境中打开该对话框，可以在 SIMULINK 窗口中选择“File：Preferences”菜单，出现如图 3.6 所示的对话框。

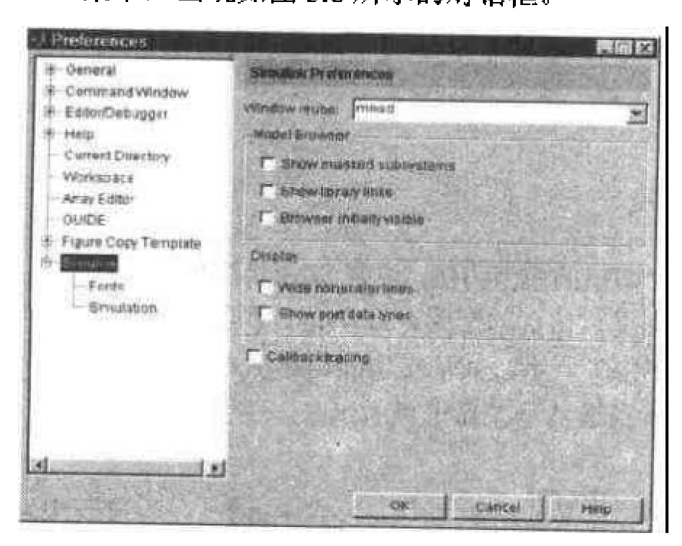


图 3.6 MATLAB 环境设置对话框

3.5.2 SIMULINK 环境的设置

SIMULINK 环境的设置如图 3.6 所示，各部分的含义如下：

- Window reuse

该栏用来设置 SIMULINK 是用现在的窗口还是打开一个新的窗口显示某个模型的子系

统。四个选项的含义分别为：

none	重新打开一个新的窗口显示子系统，当按下【Esc】键时，父窗口移至前面。
Reuse	在当前窗口中，上一层系统被子系统替代。
Replace	子系统在新的窗口中显示，而父窗口消失。当按下【Esc】键时，父窗口出现，子系统窗口消失。
Mixed	子系统在新的窗口中显示，父窗口不消失。当按下【Esc】键时，父窗口移至最前，而子系统窗口消失。

● Model Browser

定义 SIMULINK 在打开模型和子系统时，是否同时打开模型浏览器，是否封装模块中的内容以及是否显示库链接状态。

● Display

定义 SIMULINK 是否以粗线来显示非标量信号线，是否在模块方框图中显示端口数据类型。

● Callback tracing

定义 SIMULINK 在调用模块的回调例程时是否同时在命令窗口中显示回调例程的信息。

● Simulink Fonts

定义 SIMULINK 模型当中，模块与信号标示以及模型注释采用的字体。

● Solver、Workspace、Diagnostics

仿真配置的设置，包括仿真所采用的微分方程求解算法，工作空间以及诊断设置等。

3.6　一个简单的例子

3.6.1　开始

这一节我们以一个简单的模型为例，说明 SIMULINK 进行动态系统仿真的主要流程。基本步骤包括：模型的创建、仿真的配置、启动仿真和结果显示等几个部分。

该模型的框图如图 3.7 所示，其基本功能是对输入的正弦信号进行积分，然后将积分后的信号连同正弦信号本身送到示波器中显示出来。

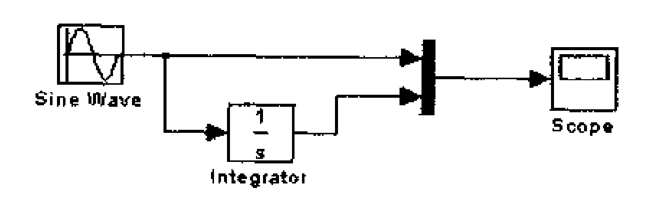


图 3.7　简单模型的方框图

仿真的第一步是启动 SIMULINK，在 MATLAB 的命令窗口输入 simulink 指令，回车后，首先出现的是 SIMULINK 的库浏览器。为了创建新的模型，单击"New Model"按钮，SIMULINK 将生成新的模型窗口。至此，创建新的模型的环境已经建立。

3.6.2 创建模型

创建模型的第一步是确定模型中包含哪些模块，在这个例中当中，模型包含四个模块，分别是正弦波模块、积分模块、示波器模块以及 Mux 模块，涉及到的模块库包括：

- Sources 库（正弦波模块）
- Sinks 库（示波器模块）
- Continuous 库（积分模块）
- Signals & Systems 库（Mux 模块）

我们可以使用库浏览器，将需要的模块从模块库中拷贝到模型窗口中。以正弦波模块为例，首先展开库浏览器中的目录树，显示 Sources 模块库，然后在库浏览器的右方选择正弦波模块，如图 3.8 所示。最后用鼠标将选中的正弦波模块拖到模型窗口中，SIMULINK 将在模型窗口中创建正弦波模块的一个副本，如图 3.9 所示。

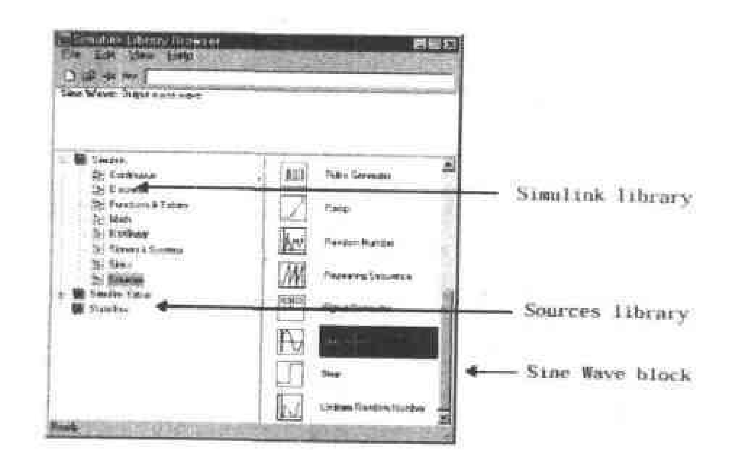


图 3.8 从库浏览器中选择正弦模块

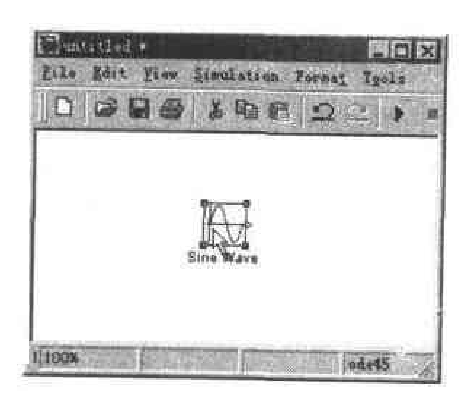


图 3.9 模型窗口中创建正弦波模块

模块在模型窗口中生成后，我们有时要调整它们的设置，使其满足系统的要求。为此，我们用鼠标双击正弦波模块，弹出图 3.10 显示的属性对话框。调整好参数后，按下“OK”按钮后，关闭属性对话框。

用同样的方法，分别在相应的模块库中将剩下的几个模块拖到模型窗口中，按照要求调整它们的位置，这时我们就完成了模块的创建，如图 3.11 所示。

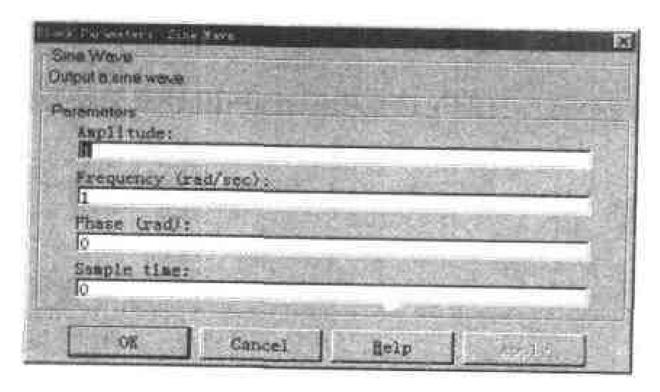


图 3.10 正弦波模块的属性对话框

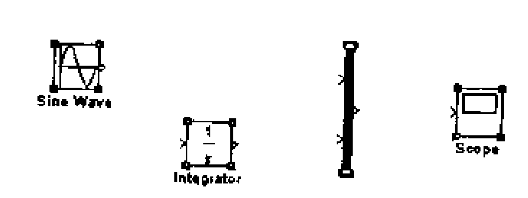


图 3.11 模块的创建

观察各个模块，可以找到模块的输入输出端口，下面我们要做的工作就是将这些模块用连线连接起来。

首先我们将正弦波模块与 Mux 模块的第一个输入口连接起来。将鼠标指向正弦波模块的输出口，光标将变成十字形，按下鼠标，并拖动到相应的 Mux 模块的第一个输入口处，注意这时的连线是虚线，如图 3.12 所示。最后释放鼠标，两个模块之间自动采用实线相连，表示两个模块已经用线连接起来。如图 3.13 所示。采用同样的方法，我们将其他的模块也用连线连接起来。这样，我们完成了整个模型的创建工作。

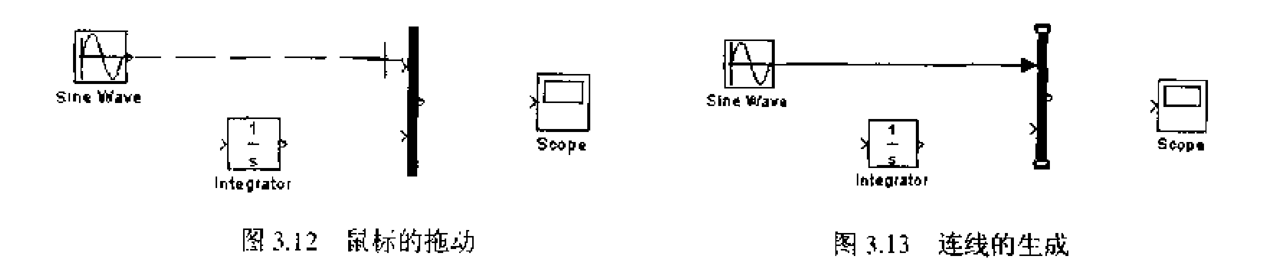


图 3.12　鼠标的拖动　　图 3.13　连线的生成

3.6.3　仿真配置

创建模型后，接下来需要对仿真的环境和参数进行配置。为此，在模型窗口中选择“Simulation：Simulation Parameters”菜单项，弹出如图 3.14 所示的配置对话框。

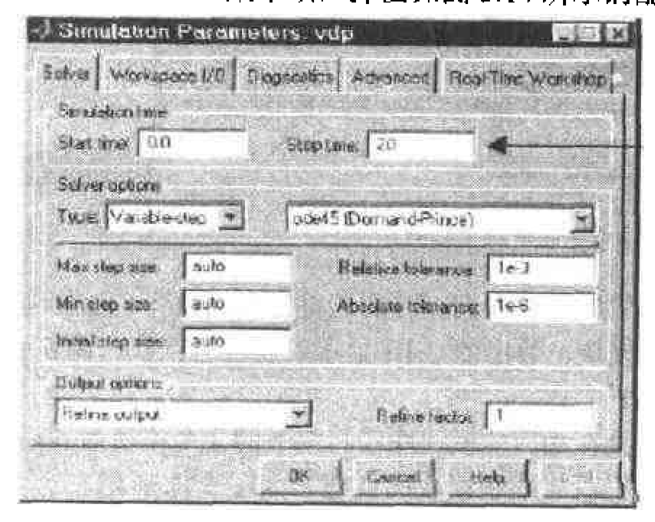


图 3.14　仿真配置对话框

在 Stop time 栏中输入 10.0，其他设置保持不变，目的是仿真模型在 0~10 秒内的动态曲线。按下“OK”按钮后，新的设置生效。

最后将模型和配置信息保存，选择“File：Save”菜单项，输入适当的模型名，这个模型将以.mdl 文件形式保存起来。

3.6.4　启动仿真

完成仿真参数配置后，接下来我们就可以启动仿真过程了。我们可以通过指令和图形两

种方式来启动模型的仿真。指令方式，以后的章节会详细介绍，我们这里采用后一种方式。在模型窗口中选择“Simulation：Start”菜单，仿真开始。仿真完成后，系统发出哔的声音提醒用户仿真结束。

3.6.5　结果

如果用户在创建模型时就打开了示波器显示窗口（通过双击示波器模块），则在仿真进行当中，我们可以看到结果的实时显示。如果示波器显示窗口没有打开，我们也可以在仿真过程或结束后随时双击示波器模块来激活示波器显示窗口，这时，示波器将以不同颜色的曲线分别显示正弦波信号以及积分后的曲线，如图 3.15 所示。

以上的过程是使用 SIMULINK 进行仿真的一般步骤，当然，我们所举的例子是很简单的，但无论多么复杂的系统模型都基本上包含以上四个步骤，所不同的是操作的复杂程度不一样。从以上我们也可以看出，采用 SIMULINK 进行动态系统的仿真，过程非常简单，大部分的工作只是用鼠标进行选择和拖动，绝大部分的操作都是可视化的，这也是 SIMULINK 受到广泛欢迎的重要原因之一。

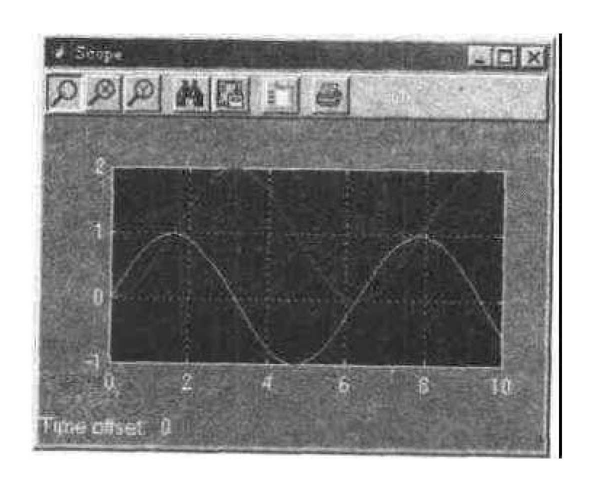


图 3.15　示波器中的仿真曲线

第 4 章　模型的创建

创建模型方框图是 SIMULINK 进行动态系统仿真的第一步。SIMULINK 为用户创建系统仿真模型提供了友好的可视化环境，用户通过鼠标的单击和拖动就能完成创建模型的大部分工作。在 SIMULINK 环境中，系统模型是由方框图表示的，模块和信号线是方框图的基本组成单位。因此，了解模块与信号线的概念和使用是创建模型的第一步。另外，SIMULINK 4.0 在 5.3 版本的基础上新增了许多功能，许多老的用户对此可能会感到比较陌生，因此在随后几节中作者将逐一进行介绍。

本章主要内容是介绍 SIMULINK 创建模型中的有关概念、相关的工具和操作方法，旨在使读者熟悉 SIMULINK 环境的使用和模型创建的基本操作，为以后进一步深入学习建立基础。

4.1　模型和模型文件

4.1.1　SIMULINK 模型的概念

SIMULINK 意义上的模型根据表现形式的不同有着不同的含义：在模型窗口中表现为可见的方框图；在存储形式上则为扩展名为.mdl 的 ASCⅡ文件；从其物理意义上讲则 SIMULINK 模型模拟了物理器件构成的实际系统的动态行为。采用 SIMULINK 软件对一个实际动态系统进行仿真，关键是建立起能够模拟并代表该系统的 SIMULINK 模型。

从系统组成上来看，一个典型的 SIMULINK 模型一般包括三个部分：输入、系统以及输出。系统也就是指在 SIMULINK 当中建立并研究的系统方框图；输入一般用信源（Source）表示，具体形式可以为常数、正弦信号、方波以及随机信号等，代表实际对系统的输入信号。输出则一般用信宿（Sink）表示，具体可以是示波器、图形记录仪等。无论是输入、输出还是系统，都可以从 SIMULINK 模块中直接获得，或由用户根据实际需要采用模块库中的模块组合而成。

当然，对于一个实际的 SIMULINK 模型而言，这三种结构并不是都必须的，有些模型可能不存在输入或输出部分。

4.1.2　模型文件的创建和修改

模型文件是指在 SIMULINK 环境当中记录模型中的模块类型、模块位置以及各个模块相关参数等信息的文件，其文件扩展名为.mdl。换句话说，我们在 SIMULINK 当中创建的模型是由模型文件记录下来的。在 MATLAB 环境中，我们可以创建、编辑并保存创建的模型文

件。

1. 创建新的模型

创建新的模型，即打开一个名为 untitled 的空的模型窗口。具体方法有：

- 单击库浏览器或模型窗口中的图标。
- 选择库浏览器或模型窗口中的菜单“File：New”。

2. 打开模型

有不同的方法用来打开已存在的模型文件。

- 库浏览器或模型窗口中的图。
- 选择库浏览器或模型窗口中的菜单“File：Open”。
- 在 MATLAB 指令窗口中键入需要打开模型的名字（不包括扩展名.mdl）。如果模型文件不在 MATLAB 的搜索路径或当前目录中，则必须指明模型的完整路径。

3. 模型的保存

SImulink 采用扩展名为.mdl 的 ASCⅡ文件保存模型。因此，模型的保存完全遵循一般文件的保存操作。

4.1.3 模型的打印

SIMULINK 模型的打印操作比较复杂，其原因在于模型本身的多层次性。

打印模型的方法既可以指令，也可以采用菜单方式。如果采用菜单方式，则选择模型窗口中的“File：Print”菜单，将出现一个打印对话框。它与 Windows 标准打印对话框的区别在于多了附加的选项框。在选项框内，用户可以设置打印的内容，包括：

- 只打印当前系统
- 打印当前系统及其上层的系统
- 打印当前系统及其下层的系统
- 打印模型中的所有系统，其中可以选择打印封装子系统和库模块的内容
- 打印每个方框图的框架

当用户选择打印当前系统时，打印对话框如图 4.1 所示，而如果用户选择打印全部模型时，打印对话框则会增加两个选项，如图 4.2 所示。

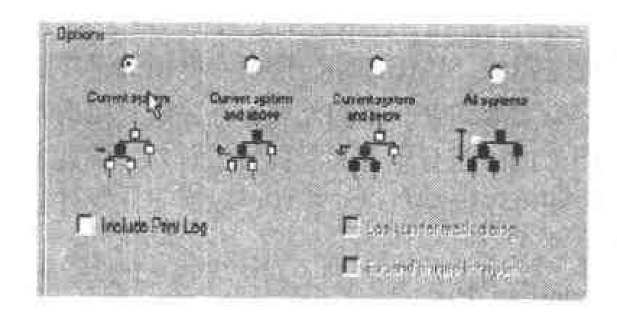


图 4.1　打印当前系统的打印对话框

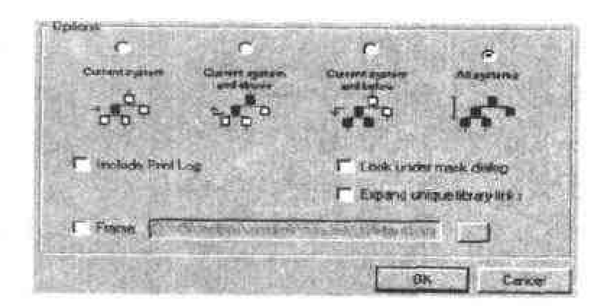


图 4.2　打印整个模型的打印对话框

4.1.4　模型的注释

在模型窗口中，书写注释的目的是帮助用户更好地理解模型，如图 4.3 所示。

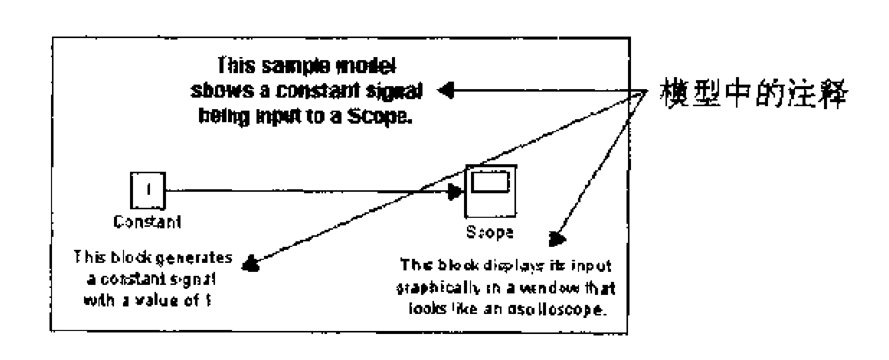


图 4.3　模型中的注释

（1）注释文本的创建

在将用作注释区的中心位置，双击鼠标左键，出现编辑框，在框中输入所需的文本后，单击编辑框以外的区域，完成注释创建。

（2）注释位置的移动

在注释文字处单击鼠标左键，待出现编辑框后，按下鼠标左键，就可把该编辑框连同其中的内容拖动到任何希望的位置。

（3）注释文字的字体控制

单击注释编辑框，再选中菜单“Format：Font”，弹出标准的 Windows 字体对话框，在该对话框中，可选择字体及文字大小。选择后，把鼠标移出注释编辑框，单击鼠标左键，即完成注释字体的修改。

4.2　模 块 操 作

4.2.1　模块的基本概念

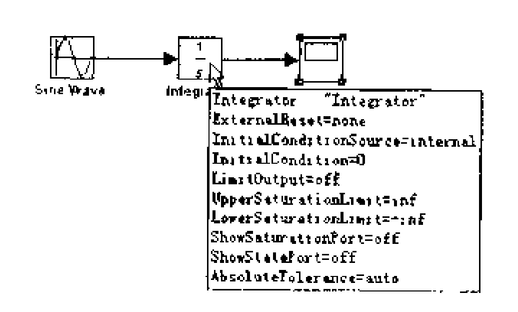


图 4.4　模块提示信息的显示

模块是组成 SIMULINK 模型方框图的基本组成单位。模块都具有一定的外观，并完成一定的功能。SIMULINK 自身包含大量的标准模块，除此之外，用户还可以定制自己的模块，以满足实际建模的需要。

1. 模块信息的显示

在 Windows 系统中，当用户将鼠标光标停留在模块图标上时，会出现一个信息提示框，其中包含了模块名、模块参数值等信息，如图 4.4 所示。用

户可以在模型窗口中选择“View：Block data tips options”菜单来激活或禁止提示信息的显示以及设置显示的具体内容。

2. 虚拟模块

SIMULINK 中的模块分为虚拟模块与非虚拟模块两种。非虚拟模块能够在模型仿真时完成特定的计算功能，而虚拟模块只有图形上的含义，其作用只是对信号的重组与管理，在仿真时不会被 SIMULINK 调用。还有的模块在这个模型中是虚拟的，而在那个模型中则是非虚拟的。这种模块我们称之为条件虚拟的。下面列举的是 SIMULINK 当中的虚拟与条件虚拟模块：

Bus Selector　　虚拟模块
Data Store Memory　　虚拟模块
Demux　　虚拟模块
Enable Port　　虚拟模块
From　　虚拟模块
Goto　　虚拟模块
Goto Tag Visibility　　虚拟模块
Ground　　虚拟模块
Inport　　虚拟模块，除非该模块位于条件子系统内或者与某个输出模块直接相连
Mux　　虚拟模块
Outport　　当该模块位于任何子系统（条件的或非条件的）中并且不在最上层模型窗口中时为虚拟的。
Selector　　除了在矩阵模式下，都是虚拟模块
Subsystem　　除了条件子系统或模块的Treat as Atomic Unit参数被选中的情况，其他都是虚拟模块
Terminator　　虚拟模块
Test Point　　虚拟模块
Trigger Port　　当输出端口不存在时为虚拟模块

4.2.2 模块的基本操作

1. 模块的选定

模块选定是许多其他操作（如复制、移动、删除）的前提操作。被选定的模块四个角处会出现黑色小矩形，称为柄（handle），如图 4.5 所示。

选定单个模块的操作方法：

用鼠标指向待选模块，单击鼠标左键，选中的模块四角出现黑色的柄。图 4.5 显示的就是一个被选中的积分模块。一旦选中一个模块，以前选中的所有模块将恢复以前不被选中的状态。

选定多个模块的操作方法：

- 方法一：按下【Shift】键，依次选定所需选定的模块。

● 方法二：按住鼠标左键，拉出矩形虚线框，将所有待选定模块包含在其中，然后松开鼠标按键，于是矩形里所有模块（包含模块之间的信号线）均被选中，如图 4.6 所示。

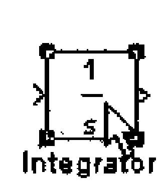


图 4.5　模块的选定

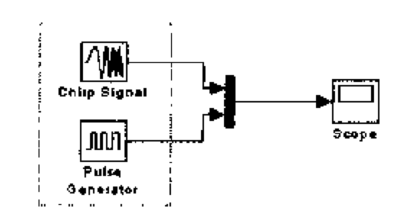


图 4.6　用虚线框同时选中多个对象

选定当前窗口中所有的模块方法：

打开窗口菜单中的“Edit”项，选择其中的“Select All”，这时，当前窗口中的所有模块都将被选中。

2. 模块的复制

模块的复制可以在模型窗口、库窗口以及模型窗口和库窗口之间实现。

不同模型窗口和库窗口之间的模块复制的方法有：

● 在一窗口中选中模块，按下鼠标左键，将它拖到另一模型窗口，释放鼠标。
● 在一窗口中选中模块，单击图标，然后用鼠标单击目标窗口中需要复制模块的位置，最后用鼠标单击图标 。

相同模型窗口内的模块复制的方法有：

● 按下鼠标右键，拖动鼠标到合适的地方，然后释放鼠标。
● 按住【Ctrl】按键，再按下鼠标左键，拖动鼠标至合适的地方，然后释放鼠标。
● 与不同模型窗口中的复制方法（B）相同。

说明：复制后所得模块具有和源模块相同的属性，并且二者同名。如果是在同一个模型中，有两个或两个以上的相同模块，那么这些模块将以其名字后面加上相应的数字加以区分。如图 4.3 所示。也可以通过复制操作将一个模块插入到一个与 SIMULINK 兼容的应用程序中去，如 Word 字处理程序。

3. 模块的移动

SIMULINK 采用一个不可见的 5 个像素的网格来简化模块的移动。模型中的所有模块和网格中的一条线对齐。用户也可采用上、下、左、右 4 个箭头来慢速移动一个模块的位置。

用户可以用 Copy、Cut、Paste 命令来拷贝或移动模块，这时只是它们的图形是被拷贝过来的，而其参数不进行拷贝。

在一个窗口把一个以上的模块拷贝到另一个窗口，其方法和拷贝一个模块是类似的，惟一的区别是在选择模块时，按下【Shift】键。

在同一个模型窗口中把一个模块从一个地方拷贝到另一个窗口，其方法是：首先选中该模块，然后拖动鼠标指针到一个用户所希望的位置，最后释放鼠标按钮。SIMULINK 将自动重新布置连到该模块上的信号线。

要移动一个以上的模块（包括它们之间的信号线），其方法如下：

（1）选中用户所需要移动的模块和连线。

如果用户用逐个法来选中一个以上的对象，在选中最后一个对象以后，不要释放鼠标。否则当用鼠标单击一个已被选中的对象时，将导致那个对象不在选中之列。

如果用户用定义方框的方法来选中一个以上的对象，用户只需用鼠标单击任何一个已经选中的模块，那么在拖动该模块的同时，就等于拖动方框内的所有对象。注意不要用鼠标单击方框内的信号线，否则的话，将只有那条信号线被选中。另外，如果在模型窗口中任何对象未曾占有的区域内单击一下，那么所有的对象就不在被选中。

（2）拖动选中的模块和连线到你所希望的位置，然后释放鼠标。

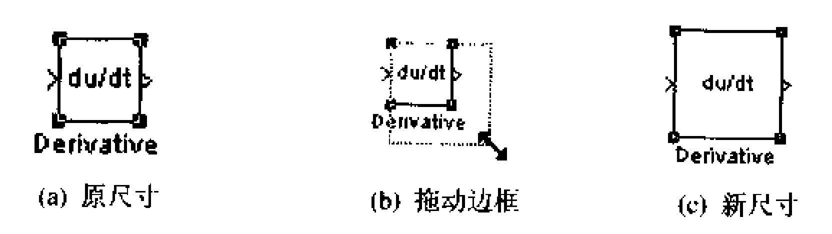
4. 模块的删除

在选中要删除的模块后，可采用以下任何一种方法删除：

- 单击键盘上的【Delete】键。
- 单击工具图标，将选定的对象剪除并放在剪贴板上，以后可以将它从剪贴板上粘贴到模型窗口中。

5. 改变模块的大小

首先选中需要改变大小的模块，然后用鼠标拖动该模块的任何一个柄。一个模块最小可定义为一个 5 x 5 的像素，而最大只受计算机屏幕的限制。当用户改变一个模块大小的时候，一个形状如箭头的标志表示所拖动的角和所拖动的方向。当模块被重新定义大小的时候，出现一个矩形虚框表示改变后的大小。这个过程如图 4.7 所示。



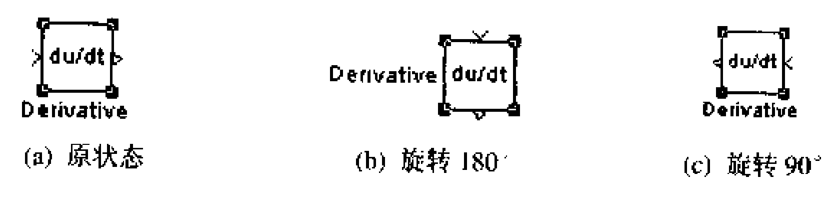
(a) 原尺寸 (b) 拖动边框 (c) 新尺寸

图 4.7 改变模块的大小

6. 改变模块的方向

SIMULINK 在默认情况下，信号总是从模块的左边流进，从模块的右边流出，即输入在左边，输出在右边。如图 4.8(a)所示。用户可以采用下面的方法来改变模块的方向：

- 单击菜单“Format：Flip Block”，可以将模块旋转 180° 。如图 4.8(b)所示。
- 单击菜单“Format：Rotate Block”，可以将选定的模块旋转 90° 。如图 4.8(c)所示。



(a) 原状态 (b) 旋转 180° (c) 旋转 90°

图 4.8 改变模块的方向

7. 模块名的操作

在一个 SIMULINK 模型当中，所有的模块名都必须是惟一的，并且至少含有一个字符，SIMULINK 在默认情况下，如果一个模块的端口在右边的情况下，那么它的名字就在它的下方；如果模块的端口在其上方或下方，那么它的名字就在它的右边。用户可以改变模块名的位置和内容。

修改模块名：单击需要修改的模块名，在模块名的四周将出现一个编辑框，可在编辑框中完成对模块名的修改。修改完毕，单击编辑框以外的区域，修改结束。

模块名的字体设置：单击菜单“Format：Font”，打开字体设置对话框，设置相应的字体。

模块名的移动：单击菜单“Format：Flip Name”，可见模块名移动到原来位置的对侧。另一种方法是单击模块名，出现编辑框后，用鼠标拖动编辑框至需要的位置。

模块名的隐藏：选中模块名，单击菜单“Format：Hide Name”，可以隐藏模块名，重新单击菜单“Format：Show Name”，又可显示模块名。

8. 模块的阴影效果

单击菜单“Format：Show Drop Shadow”，可以给选中的模块加上阴影效果。重新单击“Format：Hide Drop Shadow”则可以去除阴影效果。

9. 断开模块的连接

为了断开与某个模块相连的所有连线，按下【Shift】键不动，然后将该模块拖动到新的位置即可。

4.2.3　模块的向量化与标量扩展

1. 模块的向量化

SIMULINK 中几乎所有的模块即可以输入标量，也可以输入向量，有的还可以输入矩阵信号。向量化模块输入量和输出量之间的关系必须符合数学规则的向量或矩阵运算关系。所有向量或矩阵的大小必须相同，如图 4.9 所示。

2. 标量的扩展

标量扩展（Scalar Expansion）是把一个标量变成一个具有相同元素的向量。SIMULINK 可以对模块的输入和参数进行扩展。

（1）输入的标量扩展

当某个模块（如 Sum）具有一个以上的输入时，用户可以把向量输入和标量输入混合起来，在这种情况下，那个标量输入信号就要进行标量扩展，形成一个具有和向量输入信号维数一样的具有相同元素的向量。

例如：假设求和模块有 2 个输入端，一个输入端是向量形式[1 2 3]，另一个输入则是标量 4。该模块执行功能的数学表达式为：[1 2 3]+4＝[5 6 7]。这里，求和模块的第二个输入被扩展成与第一个输入具有相同元素的向量。仿真框图如图 4.10 所示。

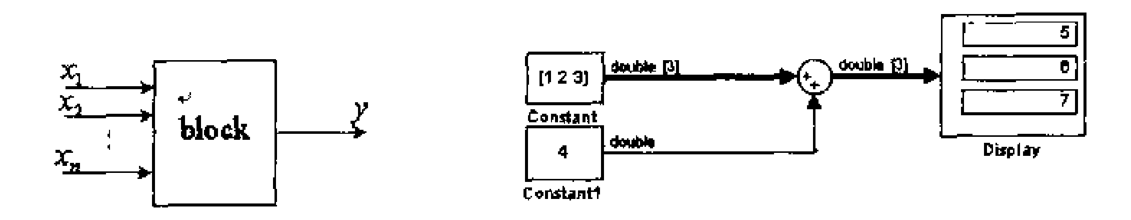


图 4.9　向量化的模块　　　　图 4.10　输入的标量扩展

（2）参数的标量扩展

对于可以进行参数扩展的模块，其参数既可以是标量，也可以是向量。当定义为一个向量参数时，向量参数中的每一个元素与输入向量中的每一个元素相对应。当定义为一个标量参数时，SIMULINK 就对标量参数进行标量扩展，自动形成具有相应维数的向量。

例如，假设“增益”模块有一个输入向量[1 2 3]。该模块执行的数学表达式为：[1　2　3]×2＝[2　4　6]。相应的 SIMULINK 方框图如图 4.11 所示。

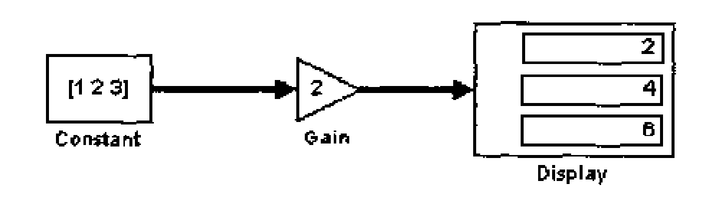


图 4.11　模块参数的标量扩展

4.2.4　模块的参数设置

几乎所有的模块都有可以进行属性设置的对话框。用鼠标双击模块，或者选中一个模块，单击菜单“Edit：Block Propertied”，SIMULINK 将打开一个模块的基本属性对话框。在该对话框中，用户可以对功能描述（Description）、优先级（Priority）、标签（Tag）、打开函数（Open Function）、属性格式（Attributes Format String）等基本属性进行设置。

图 4.12 显示的是连续系统传递函数模块（Transfer Fcn Block）的属性对话框。

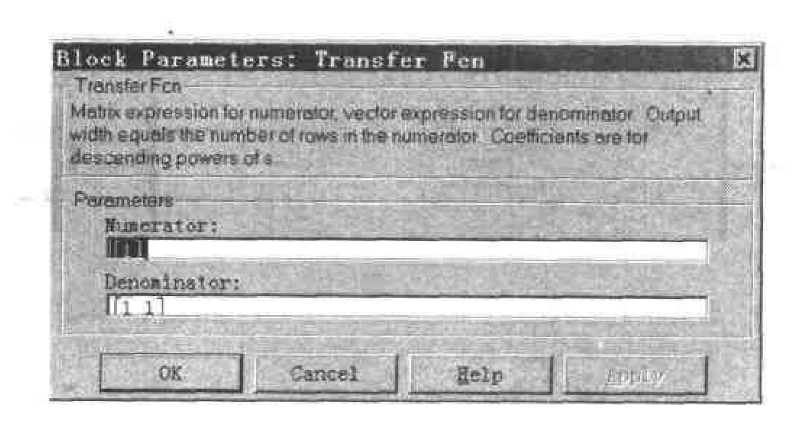


图 4.12　Transfer Fcn 模块的属性对话框

4.3　模型中的信号

4.3.1　概述

信号是 SIMULINK 在仿真过程中模块的输入、输出的数据流。这里的信号与物理世界的信号不同，SIMULINK 中的信号表示的是模块之间的逻辑关系，信号在信号线中的传输并不消耗额外的时间。

在 SIMULINK 当中，信号可以是 1 维向量，也可以是 2 维矩阵。只有一个元素的向量我们又称为标量，而矩阵中的行与列都可以看作是 1 维向量，分别称为行向量与列向量。

不同的模块接受或输出的信号维数也可能不同，有的模块可以接受所有维数的信号，而有的模块只支持标量或向量。用户在使用模块的时候，应该注意信号维数的不同。

虚拟信号指的是在仿真过程中并没有本身的含义，而实质上代表的是其他模块的信号。虚拟信号常常由虚拟模块产生，如 Mux 模块和子系统模块等。象虚拟模块一样，虚拟信号可以将众多的非虚拟信号合并成单个的虚拟信号，如 Mux 模块就具有合并多个输入的作用，这样可以简化整个模型的表示。

要注意的是，虚拟信号只有图形上的意义，并没有本质上的含义，因此 SIMULINK 在仿真过程中将会忽略模型中的虚拟信号。当我们运行或更新模型时，SIMULINK 将会将虚拟信号实际代表的其他模块的信号标示出来，这种过程称之为信号的传递。

例如图 4.13 显示的模型，驱动 G1 模块的信号 s4 就属于虚拟信号，它实际上代表的是信号 s1，同样，驱动 G2 模块的信号 s5 也是虚拟的，它实际上代表的是信号 s2。设置 SIMULINK 中的 Show Propagated Signals 选项，将会使 SIMULINK 在虚拟信号的右边标明它实际代表的非虚拟信号，如图 4.13 所示。

用户还可以通过 Mux 和 Demux 模块来创建信号总线，如图 4.14 所示。信号总线是代表一束信号的虚拟信号。它将大量的输入信号集中在一起，从而使大量纷乱的信号线变得清晰易读。通过选择“Format：Signal Dimensions”菜单，用户可以使信号总线显示所传递的信号数量。

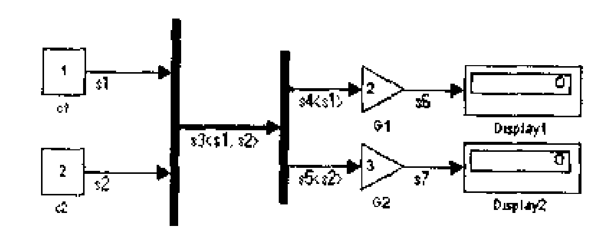


图 4.13　虚拟信号的显示

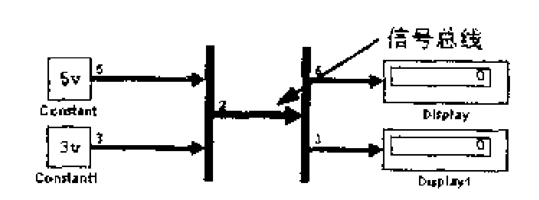


图 4.14　模型中的信号总线

4.3.2　确定输出信号的维数

SIMULINK 当中的许多模块能够产生非标量信号，如果该模块属于 Source 模块，则输出信号的维数取决于模块的参数；否则，输出信号的维数由输入信号与模块参数共同决定。下面我们主要讲讲 Source 模块的情况。

所谓 Source 模块是指没有输入端只有输出端地模块，常在模型中用作信号源。Source 模块的输出维数只与模块参数有关。这里又分为两种情况，一种是 Interpret Vector Parameters as 1-D，也就是将向量参数作为一维数组，而另一种就是相反的情况。以常数（Constant）模块为例，下面给出的就是不同情况下的输出信号维数。

常数值	Interpret vector parameters as 1-D	输出
二维标量	off	二维标量
二维标量	on	一维标量
1×N矩阵	off	1×N矩阵
1×N矩阵	on	N元素向量
N×1矩阵	off	N×1矩阵
N×1矩阵	on	N元素向量
M×N矩阵	off	M×N矩阵
M×N矩阵	on	M×N矩阵

·对于非Source模块，其输出信号的维数在经过标量扩展后将与输入信号的维数相同。

4.3.3　信号属性的设置

信号本身同样具有属性，为了观察和设置信号的属性，用户可以选择“Edit：Signal Properties”菜单，SIMULINK 将弹出信号属性对话框，如图 4.15 所示。



图 4.15　信号的属性对话框

各部分的含义如下：

● Signal name（信号名称）

该栏中输入的是信号的标示内容。

● Show propagated signals（显示传递的信号）

该栏只对于虚拟模块输出的信号才会出现。有以下几种选项：

off　不在虚拟信号的标示中显示其实际代表的非虚拟信号。

on　在虚拟信号的标示中显示其实际代表的非虚拟信号。例如，如果 虚拟信号 s1 实际代表的是非虚拟信号 s2，s3，则 s1 的标示变成 s1<s2, s3>。

all　在虚拟信号的标示中显示其间接或直接代表的所有非虚拟信号，例如，如果虚拟信号 s1 代表的是非虚拟信号 s2 和虚拟信号 s3，而 s3 又代表了非虚拟信号 s4，s5，则 s1 的标示变成 s1<s2,s4,s5>。

● Description（描述）

该栏中输入的是对信号简单的描述信息。

● Document link（文本链接）

在该栏中可以输入信号的文本信息，例如用户可以输入下列的指令：

web(['file:///' which('foo_signal.html')])

则当用户单击信号标示时，SIMULINK 将打开缺省的 Web 浏览器浏览 foo_signal.html 的 web 文件。

● Displayable（可见性）

如果用户想让模型在仿真时显示信号标示，就可以勾选此项。

● RTW storage class 和 RTW storage type qualifier

这两项只与 Real-Time Workshop 工具的使用有关，因此，如果用户不是准备用实时工具从模型中创建可执行代码，那么可以忽略它们。

4.4 信号线操作

4.4.1 绘制信号线

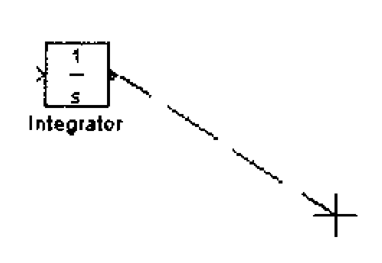


图 4.16　绘制倾斜的信号线

SIMULINK 中模块之间一般用线连接起来，称之为信号线（Signal lines）。各个模块的信号总是通过这些信号线携带并传播的。在 SIMULINK 当中，无论哪个模块都是由输入端口接受信号，由输出端口发送信号。

用鼠标在模型窗口中进行拖动即可完成信号线的绘制。具体方法是：先将光标指向连线的起点，一般为某个模块的输出端，鼠标光标将以十字显示，按下鼠标，并拖动到终点，一般为另一个模块的输入端，然后释放鼠标。SIMULINK 将根据起点和终点的位置，自动完成两个模块之间的连线。这时的连线一般由水平或垂直指向组成。

如果要绘制倾斜的信号线，可以先按下【Shift】键，然后在用鼠标进行拖动，如图 4.16 所示。

4.4.2 信号线的移动与删除

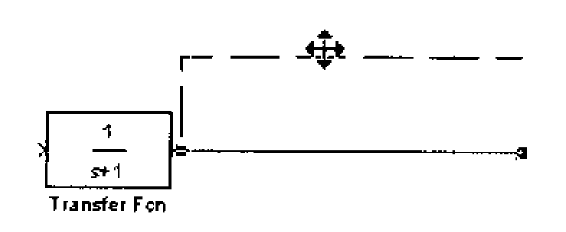


图 4.17　信号线的移动

可采用下面的方法移动某个信号线：

选中待移动的信号线（信号线端口或转折处将出现黑色的小方块），将鼠标指向它，按下鼠标左键，拖动鼠标至希望的地方，然后释放鼠标。如图 4.17 所示。

删除信号线的方法比较简单，选中待删除信号线，按下【Delete】键，或单击窗口菜单中的“Edit：Delete”。

4.4.3　信号线的分支

在实际模型中，某个模块的信号经常需要同不同的模块进行连接，这时，信号线将出现分支。绘制分支线的步骤如下：

在信号线上需要分支的某点按下鼠标右键，光标将变成十字，拖动鼠标，可以看到分支信号线自动产生，拖动到终点时，释放鼠标，完成了一条分支线的绘制，并在分支处显示出一个粗点，表示这里是相连的，如果没有出现这个粗点，则表示两条信号线相互交叉但互不相联，如图 4.18 所示。

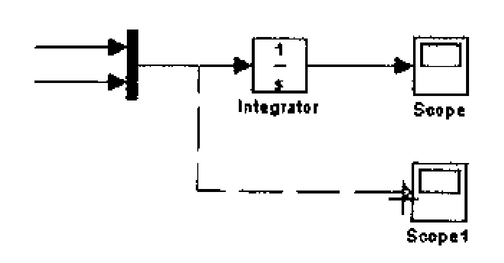


图 4.18　信号线的分支

在实际创建模型时，有时需要使两模块间的信号线转向。这种过程又称为“折曲”。实现“折曲”的方法是：选中一条信号线，将光标指向需要“折曲”的地方，按住【Shift】键，再按下鼠标左键，拖动鼠标到合适的地方，然后释放鼠标，如图 4.19 所示。

当用户选中一条信号线时，信号线的两端和弯折处会出现黑色的小方块，用鼠标拖动这些小方块，可以改变信号线的形状，这个过程称为折点的移动，具体方法是：选中已存在的信号线，将光标指向某个折点，光标将变成一个小圆圈，按下鼠标左键并拖动鼠标到合适的地方，然后释放鼠标，如图 4.20 所示。

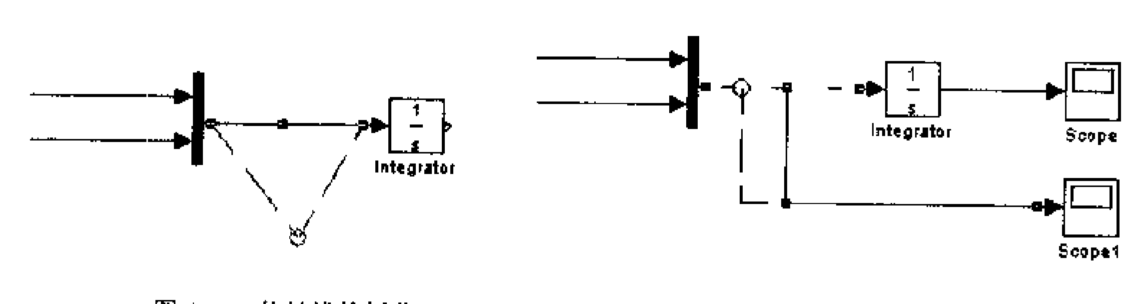


图 4.19　信号线的折曲　　　　　　　　图 4.20　折点的移动

4.4.4　信号线的显示属性

在 SIMULINK 模型方框图中，根据传递的信号性质不同，信号线也会呈现出不同的状态，例如不同的宽度和颜色。

为了区分不同信号线所传递的信号的长度，SIMULINK 可以自动用粗线显示向量信号线，并且在信号线附近的适当位置用数字标明向量的长度。具体方法是：单击菜单“Format：Wide Vecter Lines”和“Format：Vecter Lines Widths”。

SIMULINK 在创建的离散系统模型允许存在多个不同的采样频率，为了能够区分不同采样频率的模块及信号线，可选择“Format：Sample Time Color”。这样，SIMULINK 将采用不同的颜色显示不同采样频率的模块和信号线。默认情况下，黑色表示连续信号经过的模块和信号线，而红色标明最高采样频率。

4.4.5　注释信号线

添加注释：双击需要添加注释的信号线，弹出文本编辑框。输入结束后，用鼠标单击编辑框以外的地方，即完成注释的输入。

注释的修改：单击需要修改的注释，在注释四周出现编辑框，在编辑框中可以对注释进行修改。

删除注释：单击注释，出现编辑框后，双击注释，这样整个注释都被选中，按下【Delete】键可删除整个注释。

注释的复制：单击注释，待编辑框出现后，将光标指向注释，按下鼠标右键，或按下【Ctrl】的同时，按下鼠标左键，拖动鼠标到新的注释出现的地方，然后释放鼠标。

注释的传播：SIMULINK 模块中，诸如 Demux、Mux、Goto 和 From 等模块具有传播信号线注释的功能。这种功能可以使用户容易看出方框图中的信号的走向。具体方法是：将需要经过传播获取注释的信号线添加一个小于号（“<”）的注释（首先双击信号线），然后选择菜单“Edit：Update Diagram”。经过传播得到的信号线分别以大于号（“>”）和小于号（“<”）结束。如图 4.21 所示。

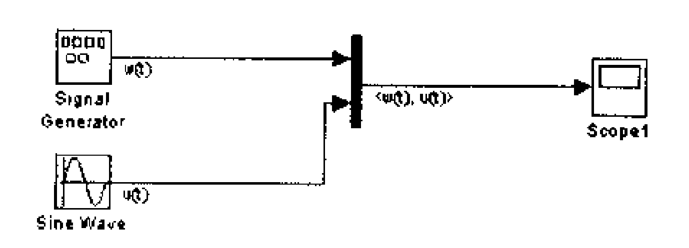


图 4.21　信号线注释的传播

4.5　数据类型与数据对象

4.5.1　SIMULINK 中的数据类型

数据类型（data style）是计算机内部表示数据的形式，它决定了为数据或变量分配的内存大小。为了能够让用户自主选择计算所用的数据类型，以优化MATLAB程序的执行效率，MATLAB允许用户定义变量的类型。同样，用户在SIMULINK环境中也可以定义信号和模块参数的数据类型。

为模型的信号和模块参数指定不同的数据类型，有时可以提高程序的运行效率。这点尤其在设计实时控制程序时有用。例如，SIMULINK允许用户定义最适合的数据类型来表示模型中的信号和模块中的参数，这样经过优化的执行代码可能效率更高，而尺寸却变得更小。

SIMULINK 在仿真开始前和仿真过程中都会严格检查所有变量和数据，确保它们是类型安全（typesafe）的，也就是说，当用户采用实时工具，如 Real-time Workshop 将该模型转化

成的执行代码不会在执行过程中产生数据溢出，从而导致错误的结果。SIMULINK 缺省情况下采用的双精度（double）数据类型能够完全保证这一点，因此用户一般情况下不必修改数据类型的缺省设置。除非修改成其他的数据类型能够很大的提高系统的执行效率，如减小执行代码的尺寸等等。

SIMULINK 使用 MATLAB 本身内建（built-in）的数据类型，也就是缺省情况下定义的类型。具体包括：

double	双精度浮点类型
single	单精度浮点类型
int8	8位符号整型
uint8	8位无符号整型
int16	16位符号整型
uint16	16位无符号整型
int32	32位符号整型
uint32	32 位无符号整型

除了内建数据类型，SIMULINK 同时允许用户定义 boolean（布尔）类型的变量，它相当与 8 位无符号整型。

在 SIMULINK 环境中，所有的模块都支持缺省的双精度浮点类型，有些模块支持 boolean 类型的输入，部分模块支持多重数据类型。

用户可以采用形如 type(value)的指令为模块的参数定义指定的类型。其中，type 是指定的数据类型，value 是模块的参数值。例如：

single(1.0)	定义大小为1.0的单精度浮点值。
int8(2)	定义大小为2的8位整型值。
int32(3+2i)	定义一个实部和虚部都是 32 位整型的复数。

用户可以在模型中自动显示模块的数据类型，具体操作为选择 Format：Port Data Types 菜单。如果用户更改了某个模块的参数类型，可以按下 Ctrl+D，更新相应模块的显示。

4.5.2　数据对象概述

SIMULINK 4.0在仿真建模过程中采用了面向对象的编程方法，这点可以通过数据对象（data objects）的使用得到充分体现。

SIMULINK 中的数据对象允许用户定义模型当中所用到的数据信息。例如规定某个参数的最大最小值，这些设置可以随着模型本身一同保存和载入，从而可以让用户创建自约束的模型。因此，数据对象的使用增强了 SIMULINK 的建模功能。

我们所说的数据对象实际上是某个数据类（data object class）的实例。数据类定义了类成员的属性和创建或操作成员变量的方法。SIMULINK 内建了两种数据类，即 Simulink.Parameter 和 Simulink.Signal，它们分别定义了参数和信号的数据对象。

（1）数据对象属性

数据对象的属性定义了对象描述的各个数据项的属性。每一个属性都是由名字和相应的数值组成的。其中的数值根据属性的不同可以是数组，也可以是某种数据结构。

（2）数据对象包

SIMULINK 将一些相关的数据内组织在一起，这样形成的结构称为包（packages）。包是比数据类更高层次上的对象。SIMULINK 自带的包称为 Simulink，我们上面提到的 Simulink.Parameter 和 Simulink.Signal 数据类，都是属于 Simulink 包。除此之外，用户还可以定义其他的包和其中包含的数据类。

用户在指令或M文件中应用数据类的方法很简单，只要同时给出包的名称和对应的数据类名。例如我们要访问Simulink包中的Parameter类，就可以输入

Simulink.Parameter

需要注意的是，两个不同的包可以包含相同名字的类，而实际上这两个数据类是完全不同的，也就是说，A包和B包都可以包含C类。从这里我们也可以看出，包这种结构可以完全避免创建类时发生名字冲突，如，用户可以创建自己的Parameter和Signal类，而不用担心它们与SIMULINK自身的类相冲突。

> 注意：包名和数据类名都是区分大小写的，因此 A.B 和 a.b 指的不是同一个数据类。

4.5.3　创建数据对象

用户可以使用SIMULINK数据浏览器（Simulink Data Explorer）或MATLAB指令来创建数据对象。我们这里首先介绍指令的用法。

使用下面的语法来创建数据对象：

h = package.class(arg1, arg2, ...argn);

其中，package和class分别是包名和类名。arg1, arg2, ...argn是传递给类构造器的参数。Simulink.Parameter 和Simulink.Signal两个标准类的构造不需要任何参数。例如，创建一个Simulink.Parameter类的实例，可以输入：

>>hGain = Simulink.Parameter;

它生成一个 Simulink.Parameter 类的实例，并将对象句柄放在 hGain 变量中。

（1）访问对象属性

用户同样可以采用SIMULINK数据浏览器访问数据对象的属性。如果采用指令方式，可以输入：

hObject.property

其中，hObject是对象名，property为属性名。

>>hGain = Simulink.Parameter;

>>hGain.Value = 5;

上面的指令创建一个模块参数对象，并设置它的值为5。

用户还可以通过迭代方法访问属性中的域，例如gain.RTWInfo.StorageClass将返回gain参数的StorageClass属性值。

（2）调用对象方法

采用下面的语法可以调用对象中的属性。

hObject.method

或

method(hObject)

SIMULINK为数据对象定义了下面两种方法：

- get　返回某个 MATLAB 结构的属性值。
- copy　拷贝一个数据对象，将生成的数据对象句柄返回

（3）数据对象的保存和载入

像不同变量一样，用户可以将数据对象包存到mat数据文件中，也可以随时从数据文件将它们载入MATLAB的工作空间。当然，前提是数据对象的类定义必须在MATLAB的搜索路径当中。

4.5.4 在 SIMULINK 模型中使用数据对象

可以在SIMULINK仿真模型当中将参数对象作为模块参数和信号值。这样可使仿真过程和生成的代码建立在面向对象的方法基础上。

可以使用SIMULINK.Parameter类来定义模块的参数。为了达到这个目的，需要：

- 在 SIMULINK 数据浏览器或命令窗口中创建参数对象。
- 设置参数对象的 Value 属性值为用户想要定义的模块参数值。
- 设置参数对象的存储类和代码创建选项等参数。
- 在模块的属性对话框或在 set_param 指令中将模块参数设置为参数对象。
- 同样可以使用 SIMULINK.Signal class 类或其继承类来定义信号属性。为此我们需要：
- 在模型工作空间中创建信号对象的实例。
- 设置信号对象的存储类和代码创建选项等参数。
- 用该信号对象的名称作为指定信号的标示。

4.5.5 数据类的创建

创建一个新的数据类实际上是编写合适的M文件来完成类的构造和实例化。SIMULINK为用户创建自己的数据类提供了一个模板，放在matlabroot/toolbox/simulink/simdemos目录下，其中定义了一个名为UserDefined的数据类，用户可以拷贝并编辑该文件，从而创建自己的数据类。如果用户要创建包含该数据类的包，则必须为这个新的包编写M文件的构造器。

（1）包的目录结构

SIMULINK将包中的所有类按照一定的结构放在某个目录中，如图4.22所示。目录的结构不同于一般的文件目录，具体包括：

每一个包都具有自己的目录结构，称为包目录。包目录的命名规则是@PackageName，其中，PackageName为包的名称。

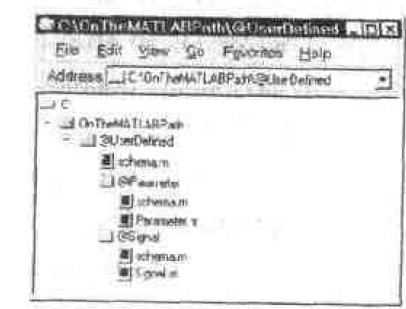


图4.22 数据对象的目录结构

包中的每一个类的代码都放在包目录的某个子目录中，称为类目录。类目录的命名规则是@ClassName，其中，ClassName为类的名称。

所有的包目录中都必须包含schema.m的M文件，它定义了包的构造器。同样，每一个类目录下也有一个schema.m文件，完成该类的构造，另外一个文件ClassName.m则完成该类的

定义。其中，ClassName为类的名称。

（2）创建一个新的包

创建一个新的包，首先在MATLAB的命令目录下创建一个新的名为@package_name的目录，其中，package_nam为新建类的名称。然后在该目录下创建schema.m文件，它实质上是一个M函数：

```
function schema ()
% Package constructor function
schema.package('PackageName');
```

（3）类的创建

在包中创建一个新的类的方法是：

- 在包目录下创建名为@ClassName 的子目录；
- 在类目录中创建一个类构造器；
- 在类目录中创建一个类的实例化函数。

（4）类构造器的创建

MATLAB通过类目录下的schema.m文件来得到类的构造器。构造器通过create_user_class函数来完成类实例的初始化工作。下面是一个简单的例子：

```
function schema()
% 类构造器函数
% 指定新建类的名称
    userClass = 'UserDefined.Parameter';
%指定父类的名称
    deriveFromClass = 'SIMULINK.Parameter';
% 调用缺省的构造器函数
% 用户定义的枚举类型
create_user_enumtype('colors', {'red', 'green', 'blue'});
% 指定该类的其他属性
addProperties = {
    'UserMATLABArray1', 'MATLAB array', []; ...
    'UserMATLABArray2', 'MATLAB array', ''; ...
    'UserDouble', 'double', 0; ...
    'UserInt32', 'int32', 0; ...
    'UserOnOff', 'on/off', 'off'; ...
    'UserString', 'string', ''; ...
    'UserColorEnum', 'colors', 'red'; ...
};
% 调用缺省的类创建函数
create_user_class(userClass, deriveFromClass, addProperties);
```

（5）实例化函数的创建

SIMULINK 通过类的实例化函数来创建类的实例。定义类的实例化函数的 M 文件名与类的名称相同。例如，如果类的名称为 Parameter，则相应的文件名为 Parameter.m，其中定义

了名为 Parameter 的函数，并且返回一个句柄。最简单的实例化函数没有任何参数，它只是调用缺省的类实例化函数，如：

```
function h = Parameter()
% Class instantiation function.
% Instantiate class
h = UserDefined.Parameter;
```

（6）数据对象函数

SIMULINK 提供了以下一些函数来完成数据对象的创建和其他操作。

create_user_class

该函数用在类的构造器中，作用是创建一个新的数据对象类。三个相关输入参数分别为：

- 新类的名称，如 UserDefined.Parameter。
- 父类的名称，如 SIMULINK.Parameter 。
- 一个指定新类属性的元胞数组。

create_user_enumtype

该函数用在类的构造器中，作用是创建一个枚举的数据类型。用户可以将它作为类属性的数据类型。其中包含两个参数：

- 枚举类型的名称。
- 一个规定该枚举类型取值范围的元胞数组。

例如，下面的指令创建一个名为colors的枚举类型。

```
create_user_enumtype('colors', {'red', 'green', 'blue'});
```

findpackage

返回包对象的句柄。例如：

```
h_SIMULINKPackage = findpackage('SIMULINK');
```

findclass

返回类的句柄。例如：

```
h_SIMULINKParameter = findclass(h_SIMULINKPackage, 'Parameter');
```

findproperty

返回一个对象属性的句柄。例如：

```
h_ParamValue = findparameter(h_SIMULINKParameter, 'Value');
```

4.5.6　SIMULINK 数据浏览器

SIMULINK数据浏览器（Simulink Data Explorer）使得用户可以显示和设置MATLAB工作空间中的变量和数据对象的值。打开数据浏览器的方法是从“Tool”菜单中选择“Data explorer”菜单项，或者在MATLAB命令行中输入slexplr指令。出现数据浏览器对话框，如图4.23所示。对话框分为左右两个部分。左边为MATLAB 工作空间中定义的的变量列表。使用Filter选项可以控制显示的数据类型，例如显示所有变量还是只显示数据对象。右边显示的是左边选中变量的具体值。

用户要想创建、重命名或删除某个数据对象，可用鼠标右键在左栏中单击相应的变量名。

为了显示属性结构中的域，可以单击属性名前的“+”按钮。用户可以通过鼠标单击来

改变其中某个值。如果该值是一个字符串，在出现编辑框，用户可以直接编辑。如果该属性值是枚举类型的，则浏览器将出现下拉菜单供用户选择；如果该值是一个数组，则浏览器出现一个数组编辑器（如图 4.24 所示），用户可以在其中设置数组的维数或完成其中某个元素的修改。

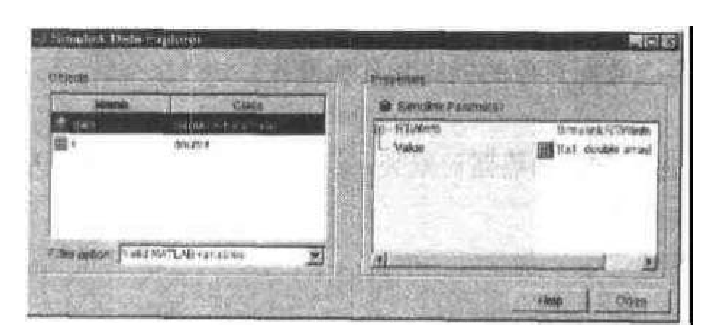


图 4.23　SIMULINK 的数据浏览器

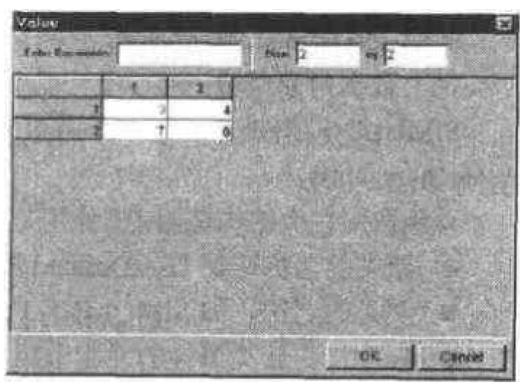


图 4.24　数组编辑器

4.6　模型创建指令介绍

4.6.1　导引

尽管SIMULINK为用户创建模型提供了可视化的图形环境，我们仍然可以通过指令来完成模型的创建。这一节我们将详细介绍相关指令的使用语法。

在使用指令完成创建模型工作以前，首先必须了解SIMULINK对象的路径管理。

- 指定模型的方法是直接使用模型文件名称（不包括扩展名 mdl），如：
  system
- 指定子系统的方法是使用子系统相对模型文件的路径加上子系统名，如：
  system/subsystem 1 /.../subsystem
- 指定模块的方法使首先给出包含该模块的模型或子系统，在加上模块名：，如：
  system/subsystem 1 /.../subsystem/block

如果模块名中包含回车键，则在回车的用“\n”代替。例如，下面的指令获取信号发生器模块中幅值参数：

```
>>cr = sprintf('\n');
>>get_param(['untitled/Signal',cr,'Generator'],'Amplitude')
ans =
1
```

如果模块名当中包含反斜杠符号（/），则可以用两个反斜杠符号（//）来与路径中的分隔符相区分。例如下面的代码获取mymodel模型当中名为Signal/Noise模块的Location参数值。

```
>>get_param('mymodel/Signal//Noise','Location')
```

4.6.2 指令详解

1. add_block

功能：在模型当中添加模块

语法：add_block('src', 'dest')

add_block('src', 'dest', 'parameter1', value1, ...)

说明：add_block('src', 'dest') 拷贝绝对路径'src'指定的模块，生成一个新的绝对路径为'dest'的模块。新模块的参数与原来模块的参数相同。可以统一用'built-in'作为SIMULINK内建（built-in）模块（不是封装库中的模块）的源路径名。

add_block('src', 'dest_obj', 'parameter1', value1, ...) 同样生成一个源模块的副本，但指定了新模块中某些参数的值。

举例：

add_block('Simulink/Sinks/Scope', 'engine/timing/Scope1')

上面的指令拷贝Simulink模型当中Sinks子系统的Scope模块，并在engine模型当中的timing子系统中生成它的一个副本，取名为Scope1。

add_block('built-in/SubSystem', 'F14/controller')

在F14模型当中创建一个名为controller的子系统。

add_block('built-in/Gain', 'mymodel/Volume', 'Gain', '4')

拷贝内建的Gain模块，并在mymodel模型当中的Volume子系统中生成它的一个副本，其中增益参数设置为4。

2. add_line

功能：在模型当中添加信号线

语法：h = add_line('sys', 'oport', 'iport')

h = add_line('sys', points)

说明：该指令在指定的模型当中添加信号线，并返回信号线的句柄。有两种方法定义信号线。

- 指定信号线相连的输入输出端口；
- 指定信号线的起始位置。

举例：

add_line('sys', 'oport', 'iport')指令在输出模块端口'oport'和输入端口'iport'之间添加一条信号线。其中'oport', 'iport'字符串为'block/port'形式，block为模块名，port为端口名。

模块的端口可以采用从上到下、从左到右的顺序进行排序，大多数模块端口都可以用它的序号来表示，如'Gain/1'或'Sum/2'，但触发、使能和状态端口必须用名字来指定，如'subsystem_name/Enable', 'subsystem_name/Trigger'，或'Integrator/State'等。

add_line(system, points)指令向模型当中添加一条信号线。Points数组的每一行指定了信号线上点的x、y坐标。坐标原点为窗口的左上角。信号线从第一行指定的点连到最后一行指定的点，如果信号线的开始点靠近某个模块的输出或其他信号线，则会自动生成与它们之间的连接。同样，如果信号线的结束点靠近某个模块的输入，也会在它们之间自动生成连接。

举例：

add_line('mymodel','Sine Wave/1','Mux/1')

在'mymodel'模型当中生成连接Sine Wave模块的输出端口与'Mux的第一个输入端口的信号线。

add_line('mymodel',[20 55; 40 10; 60 60])

在'mymodel'模型生成一条信号线段，开始与结束点坐标分别是(40,10)和(60,60)。

3．bdclose

功能：无条件关闭所有的或指定的模型窗口

语法：bdclose

bdclose('sys')

bdclose('all')

说明：不带任何参数的 bdclose 指令无条件关闭当前的模型窗口，之前对模型所做的修改不作任何保存。

bdclose('sys')指令关闭'sys'指定的模型窗口。

bdclose('all') 指令关闭所有的模型窗口。

举例：

bdclose('vdp')

关闭 vdp 模型窗口。

4．bdroot

功能：返回顶级模型的名称

语法：bdroot

bdroot('obj')

说明：不带任何参数的 bdroot 指令返回顶级模型的名称。

bdroot('obj')指令中，'obj'指指定的系统或模块的路径。返回包含指定系统或模块的顶级模型名。

举例：

bdroot(gcb)

返回当前模块的顶级模型的名称。

5．close_system

功能：关闭模型窗口或模块对话框

语法：close_system

close_system('sys')

close_system('sys', saveflag)

close_system('sys', 'newname')

close_system('blk')

说明：不带任何参数的 close_system 指令关闭当前模型或子系统的窗口。如果该窗口属于顶级模型并且已作了修改，则该窗口关闭前将询问用户是否保存修改的信息。

close_system('sys') 指令关闭'sys'指定的模型或子系统窗口。

close_system('sys', saveflag) 关闭'sys'指定的模型或子系统窗口。如果saveflag为0，系统不保存，如果为1，则系统用当前的名字进行保存。

close_system('sys', 'newname') 指令关闭指定的模型，并且使用新的名字'newname'进行保存。

close_system('blk')指令中的'blk'表示模块的绝对路径，其功能是关闭该模块的属性对话框或者调用该模块的CloseFcn 回调（callback）参数。其他的参数忽略。

举例：

close_system

关闭当前的模型窗口。

close_system('vdp')

关闭vdp模型。

close_system('engine', 1)

关闭'engine'模型，并且将模型进行保存。

close_system('mymdl12', 'testsys')

关闭'engine'模型，并且以新的名称'testsys'将模型进行保存。

close_system('engine/Combustion/Unit Delay')

关闭'engine模型当中Combustion子系统中的Unit Delay模块所关联的属性对话框。

6．detele_block

功能：从模型中删除模块

语法：delete_block('blk')

说明：delete_block('blk')指令删除指定模块，'blk'为指定模块的绝对路径名。

举例：

delete_block('vdp/Out1')

删除 vdp 模型当中的名为 Out1 的模块。

7．detele_line

功能：从模型中删除模块

语法：delete_line('sys', 'oport', 'iport')

说明：delete_block('blk')指令输出输出模块端口'oport'和输入端口'iport'之间的信号线。其中'oport', 'iport'字符串为'block/port'形式，block 为模块名，port 为端口名。

模块的端口可以采用从上到下、从左到右的顺序进行排序，大多数模块端口都可以用它的序号来表示，如'Gain/1'或'Sum/2'，但触发、使能和状态端口必须用名字来指定，如'subsystem_name/Enable', 'subsystem_name/Trigger'，或'Integrator/State'等。

举例：

delete_line('mymodel','Sum/1','Mux/2')

删除'mymodel'模型中 Sum 输出端口与'Mux 模块第二个输入端口之间的信号线。

8．find_system

功能：查询模型、模块、信号线、端口或注释

语法：find_system(sys, 'c1', cv1, 'c2', cv2,...'p1', v1, 'p2', v2,...)

说明：find_system(sys, 'c1', cv1, 'c2', cv2,...'p1', v1, 'p2', v2,...)指令使用约束 v1、v2 来寻找 sys 指定的模型或子系统，返回具有指定参数值 v1、v2 的对象句柄或路径。Sys 可以是路径名（或由路径名组成的元胞数组），句柄（或句柄向量），也可以省略不写。如果 Sys 是路径名或由路径名组成的元胞数组，find_system 指令将返回找到的所有对象路径的元胞数组；如果 Sys 是对象句柄或句柄向量，find_system 指令也将返回找到的所有对象的句柄向量；如果 Sys 忽略不写，则 find_system 指令寻找所有打开的系统，并返回路径名组成的元胞数组。

举例：

下面的列举了查询采用的约束参数：

'SearchDepth'	标量	指定查询的深度。如 0 代表只查询打开的系统；1 代表顶级模型当中的模块和子系统。缺省情况下，查询所有的系统。
'LookUnderMasks'	'none'	表示查询时跳过封装模块。
	{'graphical'}	表示查询时包括没有封装工作空间和属性对话框的封装模块。这是缺省时的设置。
	'functional'	表示查询时包括没有属性对话框的封装模块。
	'all'	表示查询所有的模块。
'FollowLinks'	'on'| 'off'	'on'表示查询过程一直连到库模块中。缺省时为'off'。
'FindAll'	'on'| {'off'}	on'表示查询范围扩展到信号线、端口和注释。注意，这时无论 sys 的数组类型是什么，find_system 指令都返回句柄向量。
'CaseSensitive'	'on'| 'off'	'on'表示查询中需要匹配查询字符串，这也是缺省值。
'RegExp'	'on'| 'off'	'on'表示查询过程中将查询表达式作为规则表达式处理。缺省情况为'off'。

举例：

>>open_bd = find_system('type', 'block_diagram')

返回所有打开的模型方框图的名称。

>>find_system('clutch/Unlocked','SearchDepth',1,'BlockType','Goto')

返回所有'Goto'模块的名字，同时要求这些'Goto'模块是 clutch 模型中 Unlocked 子系的后继。

>>gb = find_system('vdp', 'BlockType', 'Gain')

>>find_system(gb, 'Gain', '1')

在'vdp'模型中寻找增益值为1的所有增益模块，返回这些模块的名字。上面的指令也可以写成：find_system('vdp', 'BlockType', 'Gain', 'Gain', '1')。

>>sys = get_param('vdp', 'Handle');

>>l = find_system(sys, 'FindAll', 'on', 'type', 'line');

>>a = find_system(sys, 'FindAll', 'on', 'type', 'annotation');

上述的指令将返回'vdp'模型当中所有信号线和注释的句柄。

9．gcb

功能：获得当前模块的路径名

语法：gcb

gcb('sys')

说明：gcb 指令返回当前系统中当前模块的绝对路径。

gcb('sys')则返回指定系统中当前模块的绝对路径。

这里的当前模块有以下几种情况：

- 如果在模型编辑状态，当前模块就是最近选中的模块；
- 如果在包含 S 函数模块的模型仿真阶段，则当前模块就是则在执行的 S 函数模块；
- 如果是在执行回调阶段，当前模块就是正在执行其回调函数的模块；
- 如果是在解释 MaskInitialization 输入串的阶段，则当前的模块就是正在被解释的模块。

举例：

```
>>gcb
ans =
clutch/Locked/Inertia
```

返回最近选中模块的完整路径。

```
>>get_param(gcb,'Gain')
ans =
1/(Iv+Ie)
```

获取当前模块的增益值。

10．gcbh

功能：获得当前模块的句柄

语法：gcbh

说明：gcbh 指令获得当前模块的句柄。用户可以用它来区分该模块有无父系统。该指令对模块集（Blockset）的编写者尤其有用。

举例：

```
>>gcbh
ans =
281.0001
```

返回当前模块的句柄。

11．gcs

功能：获得当前系统的路径

语法：gcs

说明：gcbh 指令获得当前系统的完整路径。这里的当前系统是指：

- 如果在系统编辑状态，当前系统就是最近选中的系统；
- 如果在包含 S 函数模块的系统或子系统的仿真阶段，则当前系统就是正在执行的包含 S 函数模块的系统；
- 如果是在执行回调阶段，当前系统就是正在执行其回调函数模块的系统；
- 如果是在解释 MaskInitialization 输入串的阶段，则当前的系统就是包含正在被解释的模块的系统。

当前系统总是当前模型或当前模型的子系统。使用 bdroot 可以获得当前的模型。

举例：

```
>>gcs
 ans =
 clutch/Locked
```

12. get_param

功能：获得系统或模块的参数值

语法：get_param('obj', 'parameter')

get_param({ objects }, 'parameter')

get_param(handle, 'parameter')

get_param('obj', 'ObjectParameters')

get_param('obj', 'DialogParameters')

说明：get_param('obj', 'parameter')指令中，'obj'是系统或模块路径名。指令返回指定参数值。

get_param({ objects }, 'parameter')指令接受一个绝对路径组成的元胞数组，使用户可以得到相对应每个元胞数组元素的对象的某个属性值。

get_param(handle, 'parameter')指令返回与 handle 句柄相关的对象的属性值。

get_param('obj', 'ObjectParameters')指令返回一个描述对象属性的结构。该结构的每一个域对应与对象的某个属性。例如，Name 域对应于对象的 Name 属性。每一个参数自身又有三个域：Name、Type 和 Attributes，分别对应于指定参数的名称（如 "Gain"）、数据类型（如 "字符串"）和属性（如 "只读"）等。

get_param('obj', 'DialogParameters')指令返回由指定模块的属性对话框中所有参数名称组成的元胞数组。

举例：

```
>>get_param('clutch/Requisite Friction/Inertia','Gain')
 ans =
 1/(Iv+Ie)
```

返回 clutch 模型中 Requisite 子系统的 Inertia'模块的'Gain'属性值。

```
>>blks = find_system(gcs, 'Type', 'block');
>>listblks = get_param(blks, 'BlockType')
listblks =
'SubSystem'
'Inport'
'Constant'
'Gain'
'Sum'
'Outport'
```

该指令返回 mx+b 模型（当前模型）当中所有模块的模块类型。

```
>>get_param(gcb, 'Name')
```

返回当前选中模块的名称。

```
>>p = get_param(gcb, 'ObjectParameters');
```

>>a = p.Name.Attributes

ans =

'read-write' 'always-save'

返回当前选中模块的 Name 参数的属性值。

>>p = get_param('untitled/Sine Wave', 'DialogParameters')

p =

'Amplitude'

'Frequency'

'Phase'

'SampleTime'

得到与 Sine Wave 模块相关联的属性对话框的所有输入参数。

13．new_system

功能：生成一个空的 SIMULINK 仿真系统

语法：new_system('sys')

说明：new_system('sys')指令用指定的名称生成一个空的系统，如果'sys'是一个路径，则生成的系统是该路径中指定系统的子系统。该指令不会自动打开系统窗口。

举例：

>>new_system('mysys')

生成一个名为 mysys 的空的系统。

>>new_system('vdp/mysys')

在 vdp 模型当中生成名为 mysys 的空的子系统。

14．open_system

功能：打开某个模型窗口或模块的属性对话框

语法：open_system('sys')

open_system('blk')

open_system('blk', 'force')

说明：open_system('sys')指令打开一个指定的模型或子系统窗口。

open_system('blk')打开指定模块的属于对话框，'blk'参数是该模块的绝对路径。如果定义了该模块的 OpenFcn 回调参数，则执行相应的例程。

open_system('blk', 'force')指令打开某个封装子系统，该指令相当于单击模型窗口中的"Look Under Mask"系统菜单。

举例：

>>open_system('controller')

在缺省的屏幕位置打开名为'controller'的模型。

>>open_system('controller/Gain')

打开在'controller 模型当中 Gain'模块的属性对话框。

15．replace_system

功能：替换模型中的模块

语法：replace_block('sys', 'blk1', 'blk2', 'noprompt')

replace_block('sys', 'Parameter', 'value', 'blk', ...)

说明：replace_block('sys', 'blk1', 'blk2')指令在'sys'模型中用'blk2'类型的模块替换所有的'blk1' 模块或封装类型。如果'blk2'是内建模块，则只需要指明它的名字；如果'blk'属于另一个系统，则必须指明完整的路径。如果没有包含'noprompt参数，在替换进行之前，SIMULINK 将弹出对话框，让用户对需要替换的模块进行确认。如果包含该参数，则不会弹出相应的对话框。

replace_block('sys', 'Parameter', 'value', ..., 'blk')指令在'sys'模型中用'blk'模块替代所有具有指定参数值的模块。

举例：

>>RepNames = replace_block('f14','Gain','Integrator')

用'Integrator'模块替代'f14'模型当中所有的'Gain'模块，并将被替换模块的路径保存在 RepNames 变量中。在实际替换前，SIMULINK 将弹出对话框列出所有将被替换的模块。

>> replace_block('clutch/Unlocked','Gain','bv','Integrator')

用'Integrator'模块替代'clutch 模型的 Unlocked 子系统中所有具有 bv 增益值的模块。在实际替换前，SIMULINK 将弹出对话框列出所有将被替换的模块。

>>replace_block('f14','Gain','Integrator','noprompt')

用'Integrator'模块替代'f14'模型当中所有的'Gain'模块，但不显示对话框。

16. save_system

功能：保存某个 SIMULINK 系统

语法：save_system

save_system('sys')

save_system('sys', 'newname')

说明：save_system 指令用当前的名字保存当前的顶级模型。

save_system('sys')指令用当前的名字保存指定的顶级模型，这时，该模型应该在打开状态。

save_system('sys', 'newname') 指令用新的名字保存指定的顶级模型，这时，该模型应该在打开状态。

举例：

>>save_system

保存当前的模型。

>>save_system('vdp')

保存'vdp'模型。

>>save_system('vdp', 'myvdp')

用新的名字 myvdp 保存'vdp'模型。

17. set_param

功能：设置系统或模块参数

语法：set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)

说明：set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)指令将指定的参数设置成指定的值。obj 为该模块的路径名。

在模型仿真期间，用户还可以在工作空间中改变相应的参数值，并且用新的参数值更新模块。

具体做法是选择模型窗口中的“Edit：Update Diagram”菜单。

举例：

>>set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')

设置 vdp 模型中的 Solver'和 StopTime 仿真参数。

>>set_param('vdp/Mu', 'Gain', '1000')

将 vdp 模型中的 Mu 模块的 Gain(增益)属性设置成 1000。

>>set_param('vdp/Fcn', 'Position', [50 100 110 120])

为 vdp 模型中的 Fcn 模块设置新的尺寸和摆放位置。

>>set_param('mymodel/Zero-Pole','Zeros','[2 4]','Poles','[1 2 3]')

为'mymodel 模型中的 Zero-Pole 零极点模块设置新的零极点。

>>set_param('mymodel/Subsystem', 'k', '10')

为指定的封装子系统中的增益模块设置新的增益值。其中，k 是与该模块的增益值相关联的变量。

>>set_param('mymodel/Compute', 'OpenFcn', 'my_open_fcn')

为 mymodel 模型中的 Compute 模块设定 OpenFcn 回调参数值。当用户双击模块（打开模块）时，系统将执行 my_open_fcn 的例程。

18. simulink

功能：打开 SIMULINK 模块库

语法：simulink

说明：simulink 指令将打开或激活 SIMULINK 模块库浏览器（Windows 系统）。

4.7　模块库与连接

4.7.1　导引

模块库简称库（Library）是 SIMULINK 中十分重要的概念。通过模块库，用户可以方便地使用第三方开发提供的专业模块，并且可以创建自己的模块库。下面是与模块库相关的几个基本概念。

库（Library）是指库模块的集合。用户可以通过选择“File：New Library”菜单来创建一个新的库。

库模块（Library block）是指库中的模块。

参考模块（Reference block）指的是库模块的一个拷贝。

连接（Link）是指在库模块与参考模块之间建立起来的一种联系。这种联系使得当库模块发生变化时，对应的参考模块也随之变化。

拷贝（Copy）是指从库模块或其他参考模块生成参考模块的操作。图 4.25 显示了以上几

个概念之间的关系。

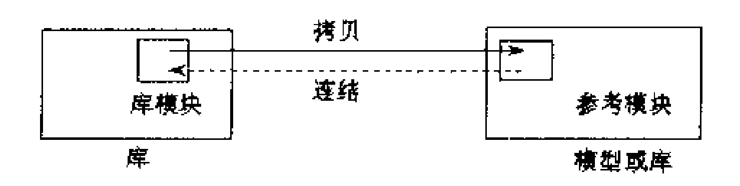


图 4.25　连接与拷贝的关系

4.7.2　库的创建与修改

有两种方法创建新的库，一种是菜单方式。在“File：New”菜单项的弹出项中选择“Library”，SIMULINK 将创建名为 Library: untitled 的新窗口。

另一种使指令方式。在命令窗口中输入：

>>new_system('newlib', 'Library')

可以创建名为'newlib'的库。然后采用 open_system 指令可以显示指定的库。在对库进行拷贝操作前，必须先保存新建的库。

当用户打开一个库时，该库处于封锁状态（Lock），因此不能对库的内容进行修改，如果用户需要解开封锁，可以选择“Edit：Unlock Library”菜单项。

4.7.3　创建对库的连接

当从库中拷贝或采用鼠标拖动模块图标到模型窗口中，就自然在模型中创建了一个到库的连接。参考模块时库模块的一个副本，用户可以改变参考模块的参数值，但不能封装该模块，如果该模块已经是封装的，也不能对它进行编辑。同时，用户不能设置它的回调函数。如果该连接是连接到一个子系统，也不能修改子系统的内容。

参考模块与库模块是通过名字相关联的。如果SIMULINK在对模型中的模块进行更新时找不到相应的库模块，或者在搜索路径中找不到原来的库，则相应模块的连接变成unresolved状态，在命令窗口中输出如下的错误消息：

Failed to find block *"source-block-name"*
in library *"source-library-name"*
referenced by block
"reference-block-path".

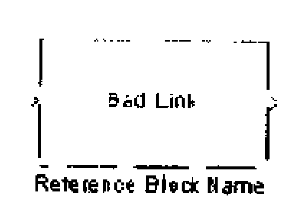


图 4.26　unresolved 状态的模块外观

同时，该模块的边框变成红色的虚线，如图 4.26 所示。

用户可以通过以下措施解决这个问题：

- 删除连接无效的模块，重新从库中进行拷贝。
- 在 MATLAB 搜索路径中创建目录，其中包含所需要的库。然后选择“Edit：Update Diagram”菜单。
- 双击参考模块，在对话框中输入模块的路径，然后单击“Apply”或“Close”按钮进

行确认。

SIMULINK 还允许用户禁止模块的连接，这样，在模型仿真过程中，SIMULINK 将忽略这些禁止的连接。为此，用户需要在“Edit”或上下文菜单中选择“Link options”，然后选择“Disable link”。选择“Restore link”菜单可以保存连接的状态。

4.7.4 修改具有连接的子系统

SIMULINK 允许用户修改具有库连接的子系统，但前提是不改变子系统本身的结构。如果用户所作的修改使子系统的结构发生改变，必须事先禁止子系统的连接。涉及子系统结构的操作包括：增减子系统中的模块或信号线、改变模块端口的数目等。模块参数的修改不会改变子系统的结构。

当用户保存一个结构发生变化的连接时，SIMULINK 将提示用户是同意这种变化，还是放弃。如果用户选择同意（propogate），SIMULINK 会对相应的库模块作相同的修改。如果用户选择放弃（discard），SIMULINK 将忽略这种变化，同时用库模块替代现在的参考模块。无论哪种结果，最终参考模块与库模块在结构上必须完全一致。

而对于具有非结构变化的连接，SIMULINK 在保存时不再提示用户进行这样的处理，但用户同样可以进行设置。为此，用户可以选中相应的模块，选择“Edit：Link options”菜单，然后选择“Propagate/Discard changes”菜单。如果用户想了解参考模块与库模块之间有何非结构的不同，可以选择其中的“View changes”菜单。

4.7.5 连接模块的更新与显示

SIMULINK 在下列情况下对模型或库中的模块进行更新：

- 模型或库被装载；
- 用户选择“Edit：Update Diagram”菜单或运行仿真；
- 使用 get_param 指令查询模块的 LinkStatus 参数值；
- 用户使用 find_system 指令时。

用户可以根据需要断开参考模块与相应库模块的连接，使得参考模块仅仅成为库模块的一个简单拷贝。库模块的变化也不再影响该模块。断开模块的连接可以让用户单独发布模型，而不需要库的支持。

为了断开某个模块的连接，首先使该模块在无用（disable）状态，然后选中该模块，选择“Link options：Break Library Link”菜单。用户还可输入下面的指令完成同样的操作：

```
>> set_param('refblock', 'LinkStatus', 'none')
```

也可以保存模型并断开所有模块与库的连接：

```
>> save_system('sys', 'newname', 'BreakLinks')
```

如果用户需要查找参考模块所连接的库模块，可以选择“Link options: Go To Library Link”菜单，这时如果相应的库已经打开，则SIMULINK直接找到相应的库模块；否则，SIMULINK首先打开库，然后选中相应的库模块。

所有的模块都具有LinkStatus的属性，不同值的含义为：

none　　模块不是参考模块；

resolved　　　　连接处于resolved状态；
unresolved　　　连接处于unresolved状态；
implicit　　　　该模块在某个连接模块的内部；
inactive　　　　连接处于无效状态。

为了清楚地观察各个模块的连接状态，SIMULINK允许用户用模块图标左下角的箭头表示该模块与相应库模块的连接，如图4.27所示。

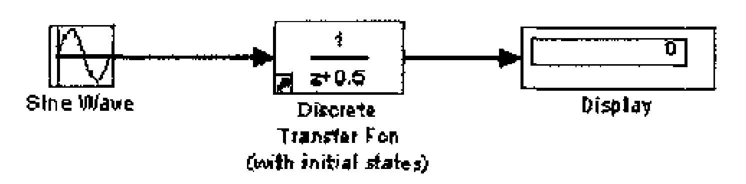


图 4.27　显示连接状态的模块

为了在模块图标中显示连接，用户需要选择“Format：Library Link Display”菜单，如果选择“User”，则SIMULINK只显示与用户定制库的连接，如果选择“All”，则显示与所有库的连接。

箭头的不同颜色显示了连接的不同状态：
黑色(Black)　　　有效的连接。
灰色(Grey)　　　无效的连接。
红色(Red)　　　　有效的连接，但进行了修改。

4.7.6　浏览模块库

通过库浏览器（Library Browser），用户可以方便地定位或拷贝库模块到模型当中。为此，用户可以单击 MATLAB 桌面上的 Library Browser 图标，或者在指令窗口中输入 SIMULINK 指令。

为了将自己创建的库加入到库浏览器中，用户必须在该库的目录中创建一个名为 slblocks.m 的 M 文件。创建该文件最简单的方法是将已经存在的其他 slblocks 文件作为模板，在此基础上进行修改。用户可以通过下面的指令寻找所有的 slblocks 文件：

```
>> which('slblocks.m', '-all')
```

确信用户创建的库存放的目录在 MATLAB 的搜索路径中，下次启动时，SIMULINK 将在搜索路径中自动搜索所有的库信息，并将它们在库浏览器中显示出来。

4.8　模型的查找与浏览

4.8.1　模型对象的查找

SIMULINK 4.0 新增了模型对象的查找工具，借助这种工具，用户可以在大量的模型当中迅速定位所要寻找的对象，包括模块、信号、状态变量等。

为了寻找指定的对象，在 SIMULINK 窗口中选择“Edit：Find”菜单，SIMULINK 将弹出图 4.28 所示的查找对话框。

用户可以在对话框中的 Filter options（过滤选项）栏和 Search criteria（查询关键字）栏输入所查对象的特征。如果用户不只打开一个模型或子系统，则可以在 Start in system 栏中选择其中一个系统作为查询的对象。最后单击“Find”按钮，SIMULINK 将符合用户输入特征的所有模块在下面的结果栏中显示出来，如图 4.29 所示。

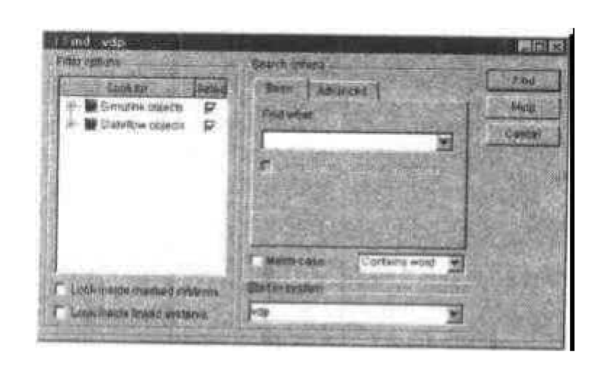


图 4.28　SIMULINK 的查找对话框

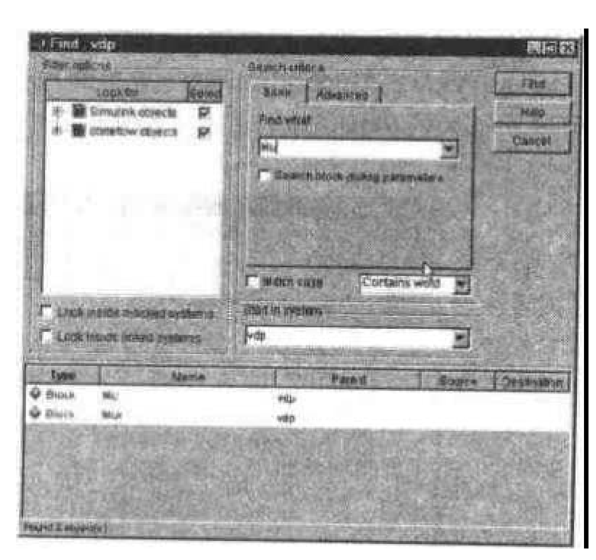


图 4.29　SIMULINK 的查询对话框

在查询的结果中双击相应的对象，SIMULINK 将自动打开包含该对象的模型或子系统，并且将查询得到的对象高亮显示。

● 过滤栏的设置

过滤栏的作用是让用户设置所查对象的类型，以及查找的方式。如是否需要查找封装子系统中的内容，或者查找与库相连的子系统中的对象。典型的过滤栏如图 4.30 所示。

● 关键字栏的设置

用户可以在该栏中输入所查模块的具体信息，其中包括 basic 和 advance 属性页。如果用户只想利用对象的名称或模块参数名作简单的查询，则只要在 basic 栏中输入需要匹配的字符串。而 advance 页则为用户提供了查询的高级功能，如图 4.31 所示，用户可以在其中设置逻辑关系完成较为复杂的查询。具体的设置方法可以参考相应的帮助文档，这里不再赘述。

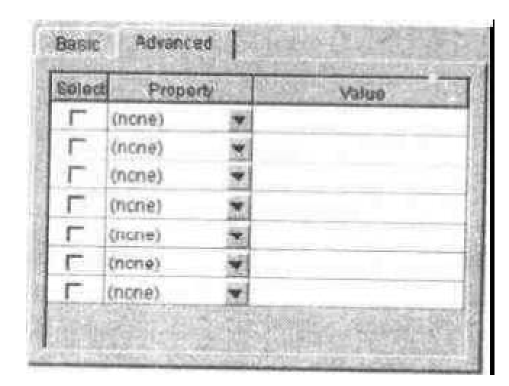


图 4.30　过滤栏的设置

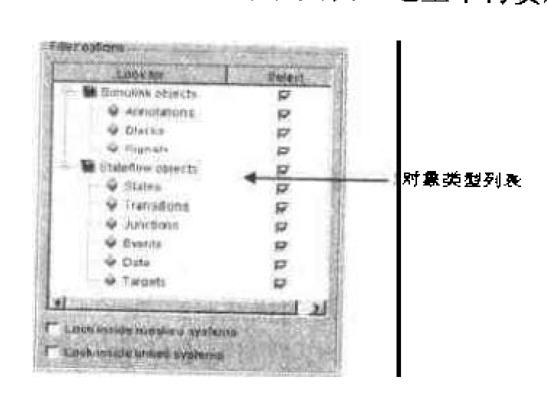


图 4. 31　SIMULINK 的高级查询功能

4.8.2　模型浏览器的使用

借助模型浏览器（Model Browser），用户可以完成：

- 按层次结构浏览某个模型；
- 直接打开模型中的系统；
- 确定模型中包含的模块；
- 使用用户的源控制系统（source control system）来管理模型。

选择“View：Model Browser”菜单即可打开模型浏览器。它由两部分组成，左边是显示模型层次结构的树状结构，右边则是模型的方框图，如图 4.32 所示。

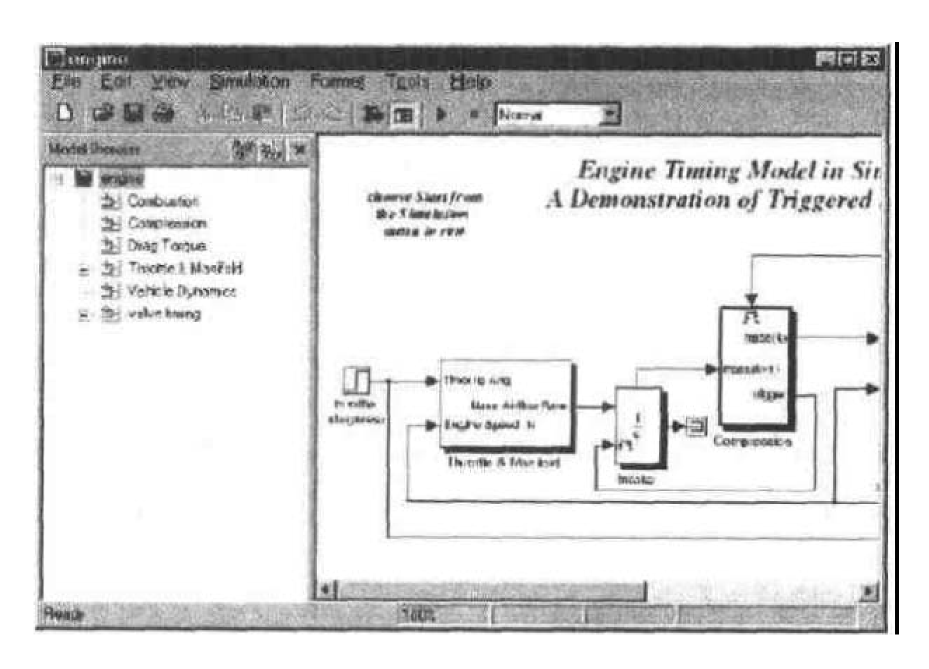


图 4.32　SIMULINK 的模型浏览器

用户可以用鼠标或键盘在左边的树目录中选择所要浏览的子系统，右边的窗口将自动显示相应的方框图。通过在 SIMULINK Preferences 中的设置，模型浏览器还可以自动显示模型当中的连接和封装子系统。

4.9　建模的方法与技巧

4.9.1　创建子系统

当创建的模型很复杂时，我们可以通过将相关的模块组织成子系统来简化模型的显示。使用子系统具有下面的优点：

- 有助于减少模型窗口中显示的模型数量；
- 允许用户将模型中功能相关的模块组织在一起；

● 使用户可以建立分层的模型框图。其中，子系统模块位于某个层次，而组成子系统的模块则位于另外的层次。

创建子系统的方法大致可分为两种：一种是在模型中加入一个子系统（Subsystem）模块，然后打开子系统窗口并进行编辑；另一种则是直接用鼠标框选组成子系统的模块，然后单击相应的菜单完成子系统的创建。具体创建方法我们在下一章详细描述。

通过子系统，用户可以创建分层的模型结构，用户可以使用模型浏览器来浏览模型中的层次关系，也可以采用下面的模型浏览指令。

Open：这个指令用来打开对应的子系统。具体操作方法是选择“Edit：Open”菜单，或按下回车键，也可以双击子系统模块。

Go to Parent：这个指令显示当前窗口中的子系统的上一层系统。具体操作方法是选择“View：Go to Parent”菜单，或按下【Esc】键。

SIMULINK 会在子系统模块上标明端口。标示的内容一般是子系统的输入或输出模块名称，子系统正是通过这些模块与子系统外的模块进行交互。用户可以通过下面的操作隐藏某些或所有的端口标示：

● 选择子系统模块，单击“Format：Hide Port Labels”菜单来隐藏所有端口标示；
● 在子系统中选择某个输入或输出模块，然后单击“Format：Hide Port Labels”菜单来隐藏相应端口标示；
● 在子系统模块的对话框中检查 Show port labels 栏。

例如，图 4.33 显示了子系统端口标示的情况。左边的方框图显示子系统具有两个输入端口和一个输出端口，右边的子系统模块标示了对应的端口。

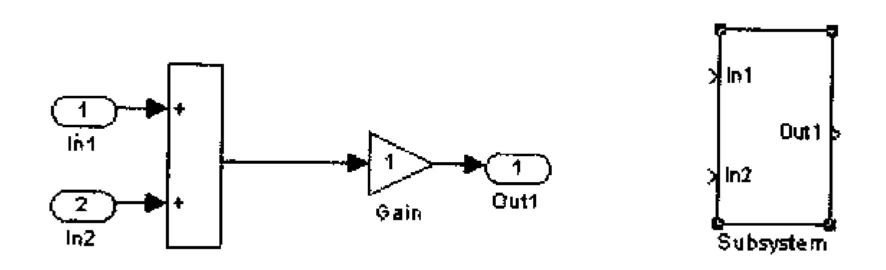


图 4.33　子系统的端口标示

SIMULINK 允许用户设置访问子系统的权限，禁止使用者观看或修改子系统的内容，这可以通过在子系统属性对话框中的 Access 栏中设置相应的参数值来实现。用户可以将它设置成 ReadOnly 或 NoReadOrWrite，其中 ReadOnly 参数表示允许使用者查看或局部拷贝子系统的内容，但禁止修改原始拷贝中的内容。NoReadOrWrite 参数表示禁止对子系统的查看、修改或拷贝。

4.9.2　使用回调例程

用户可以在 SIMULINK 当中定义与模块或模块参数相关联的 MATLAB 语句，这些语句在用户对模块进行某种操作时由 SIMULINK 自动调用，这些指令我们称之为回调例程。例如，当模型用户双击模块的名字或模块的路径改变时，SIMULINK 将自动调用与模块的 OpenFcn 参数相关联的回调例程。

回调例程的设置是通过 set_param 指令完成的。例如，执行下面的指令，则当用户在 mymodel 模型中双击 Test 模块时，变量 testvar 将被执行。

```
>>set_param('mymodel/Test', 'OpenFcn', testvar)
```

读者可以参看SIMULINK中的clutch模型（clutch.mdl），学习其中各种模块例程的定义方法。

回调跟踪让用户能够确定 SIMULINK 在打开或仿真模型时将执行的回调例程以及执行的次序。为了使用这种特性，用户可以在 SIMULINK 参考对话框中选中 Callback tracing 栏，也可以执行 set_param(0, 'CallbackTracing', 'on')指令。这样，当 SIMULINK 执行某个回调例程时，将会在命令窗口中输出其相关信息。

与模型相关的回调参数列举如下：

CloseFcn	在模块方框图关闭时执行。
PostLoadFcn	在模型载入之后执行。当模型载入后需要与用户进行交互时，该参数尤其有用。
InitFcn	在模型仿真开始时调用。
PostSaveFcn	在模型保存后调用。
PreLoadFcn	在模型载入前调用。
PreSaveFcn	在模型保存前调用。
StartFcn	在模型仿真开始时调用。
StopFcn	在仿真停止后执行。StopFcn 被执行前，输出将存储到工作空间的变量合文件中。

与模块相关的回调参数列举如下：

CloseFcn	当模块用指令close_system关闭时执行。
CopyFcn	在模块被拷贝后执行。
DeleteFcn	在模块被删除后执行。
DestroyFcn	当模块被销毁时执行。
InitFcn	在模块方框图被编译或模块参数被执行前执行。
LoadFcn	在模块方框图被载入后执行。
ModelCloseFcn	在模块方框图关闭时执行。
MoveFcn	当模块被移动或模块尺寸变化时执行。
NameChangeFcn	在模块的名称或路径发生改变时执行。
OpenFcn	在模块被打开时执行。
ParentCloseFcn	在包含该模块的子系统关闭时执行。
PreSaveFcn	在模块方框图被保存前执行。
PostSaveFcn	在模块方框图被保存后执行。
StartFcn	在模块方框图被编译或仿真开始前执行。
StopFcn	在仿真结束时执行。
UndoDeleteFcn	当撤销模块删除操作时执行。

4.9.3　建模时的考虑

以下是建模时考虑的一些因素：

（1）内存因素

一般而言，计算机的内存越大，SIMULINK运行得越好。

（2）使用分层

如果系统的模型比较复杂，使用子系统的分层机制将会大大简化模型，用户可以将主要精力放在系统信号的主要走向，而忽略其中各个功能模块的实现细节。同时，也方便模型的理解和阅读。

（3）使用注释

一般而言，具有详细的注释和帮助信息可以让模型更容易阅读和理解。因此，在模型创建过程中，应该尽可能为信号添加注释。另外，为模型添加说明信息，可以让创建的模型更容易阅读。

（4）建模策略

用户在创建模型中可以发现，如果模型中的模块重复比较多，那么模型的保存也相对容易。我们可以将反复用到的模块，全部放在新建的模块库中，这样可以在MATLAB窗口中通过模块的名字来访问它们。

一般来说，在创建模型前，可以首先纸上将系统的方框图设计好，然后将需要用到的模块放在模型窗口当中，最后再完成模块的连线。这样可以避免频繁的打开模块库。

4.9.4　方程的建模

我们知道，要仿真的系统一般是采用数学方程来描述的，因此在模型的创建过程中，我们经常遇到的是对方程的建模，而初学者往往对此感到疑惑。下面我们通过两个例子来说明方程建模的一般步骤。

1. 将摄氏温度转变成华氏温度

首先来模拟一个将摄氏温度转变成华氏温度的方程：

$$T_F=9/5(T_c)+32 \tag{4.1}$$

（1）建模所需模块的确定

在进行建模之前，首先需要确定建立上面这个模型所需要的模块：

- 一个 Gain 模块，用于定义常数增益 9/5，Gain 模块来源于 Math Library；
- 一个 Constant 模块，用来定义一个常数 32，Constant 模块来源于 Source Library；
- 一个 Sum 模块，用来将两项相加，Sum 模块来源于 Math Library；
- 一个 Sine Wave 模块，用来作为输入信号，Sine Wave 模块来源于 Source Library；
- 一个 Scope 模块，用来显示系统输出，Scope 模块来源于 Sinks Library。

（2）模块的拷贝

把上面这些模块从各自的模块库中拷贝到用户的模型窗口中，如图 4.34 所示。

分别打开 Gain 模块和 Constant 模块（双击模块图标），分别再将它们设置为 9/5 和 32。然后，单击“OK”按钮。打开 Sine Wave 模块，把它的幅值设为 10，以便得到较大的温度变

化。

（3）模块的连接

把各个模块连接起来，得到如图 4.35 所示的方框图。

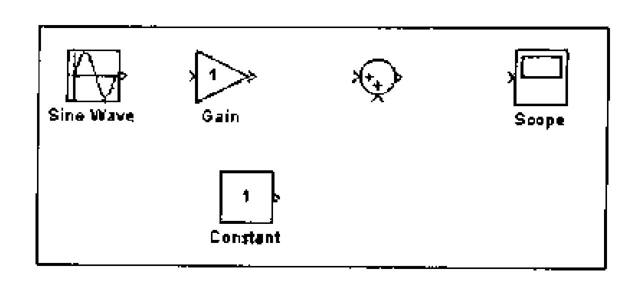


图 4.34　从库中拷贝所需模块

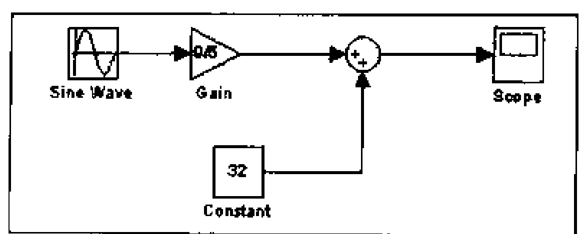


图 4.35　完成后的方框图

Sine Wave 模块代表摄氏温度，Gain 模块输出为 9/5，这个值在 Sum 模块中和 Constant 模块中的常数 32 相加后得到输出，这个输出就是华氏温度。打开 Scope 模块就可以观看这个输出值的变化曲线，其中，Scope 模块的 x 轴设为比较小的时间，如 10s，而把 y 轴设置得比幅值略大一些，以便能够得到整个曲线。

（4）开始仿真

在模型窗口得"Simulation"菜单中选择"Parameters"选项，定义 Stop time 为 10s，Maximum step size 为 0.1s，然后在 "Simulation" 菜单中选择 "Start" 命令，仿真就开始。仿真曲线如图 4.36 所示。

2. 模拟一个简单的连续系统

下面来模拟一个微分方程

$$\dot{x} = -2x + u \tag{4.2}$$

在这个模型中，需要对 $\dot{x}$ 进行积分，并产生输出，因此需要一个积分（Integrator）模块。另外需要 Gain 模块、Sum 模块。为了产生一个输入信号，可以采用 Signal Generator 模块来产生方波。除此之外，还需要一个 Scope 模块来观察模型的输出。把这些模块按第一个例子所述的方法从各自的库中拷贝到模型窗口中，并用连线连接起来，再把 Gain 模块的增益设置为 2，把信号发生器的信号类型设为方波，频率为 0.2Hz，结果得到如图 4.37 所示的方框图。

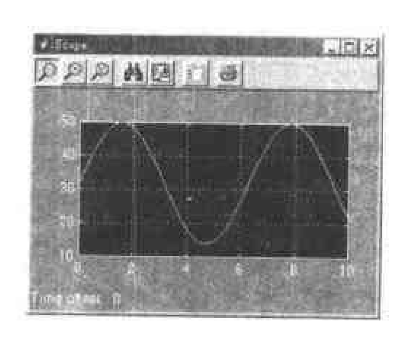




图 4.36　华氏温度的输出曲线

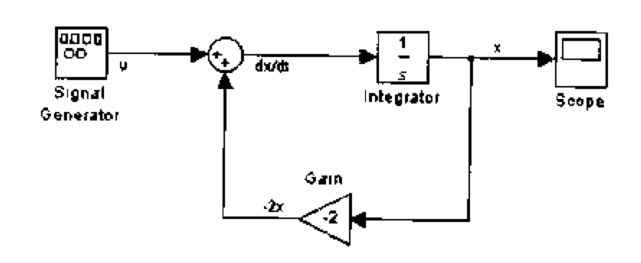


图 4.37　简单连续系统的方框图

在这个模型当中，可采用 "Format：Rotate Block" 菜单来改变 Gain 的方向。其中最重要的概念是包括 Sum 模块、Inegrator 模块的输入且依赖于 x。因此，只有构成闭环回路，才能算出 x 的值。因此 x 和 dx/dt 这种关系必须构成闭环才能实现。仿真时，每计算一步，Scope

模块就显示一步 x 的值。如果仿真时间设为 10s，得到的仿真曲线如图 4.38 所示。

在这个例子当中，所模拟的方程也可以用传递函数的形式来表示，模拟就采用 Transfer Fcn 模块，u 作为该模块的输入，x 作为该模块的输出。因为 Transfer Fcn 模块实现的系统为 x/u，因此对该微分方程进行拉氏变换，得到：

$$sx=-2x+u \tag{4.3}$$

最后，得到

$$x/u=1/(s+2) \tag{4.4}$$

即传递函数为 1/(s+2)，其中分子为 1，分母为 s+2，打开 Transfer Fcn 模块的对话框，定义相应的参数。最后得到如图 4.39 所示的方框图，运行仿真，仿真结果和前面的模型完全一样。

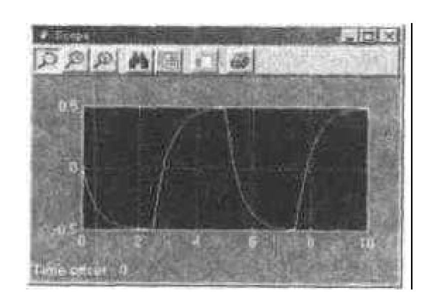


图 4.38　简单连续系统的仿真曲线

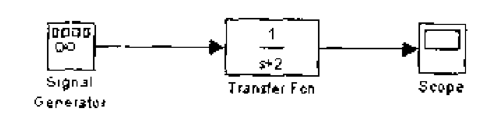


图 4.39　用传递函数模块表示的简单连续系统

4.9.5　快捷键介绍

1. 模块操作的快捷键

LMB（鼠标左键）	选择一个模块
Shift + LMB	选择多个模块
Tab	选择下一个模块
Shift + Tab	选择以前的模块
Drag block（拖动鼠标）	从另一个窗口中拷贝模块
Ctrl +LMB 和 drag；或者 RMB 和 drag	复制一个模块
LMB	连接模块
Shift +drag block	断开模块的连接
Enter Return	打开选择的子系统
Esc	返回当前子系统的上一层系统

2. 信号线操作的快捷方式

LMB	选择一条信号线
Shift +LMB	选择多个信号线
Ctrl + drag line；或 RMB +drag line	绘制信号线的一条分支线
Shift + drag line	绘制一段信号线

3. 信号标示的快捷方式

双击信号线	进行相关信号标示的输入

Ctrl + drag label	拷贝信号标示
Drag label	移动信号标示
单击标示文本	对标示内容进行编辑
Shift + 单击标示，然后按下【Delete】键	删除信号标示

4. 模型注释的快捷方式

双击方框图的空白处	输入模型注释
Ctrl + drag label	拷贝注释
Drag label	移动注释
单击注释文本	对注释内容进行编辑
Shift + 单击注释文本，然后按下【Delete】键	删除模型注释

4.10 管理模型的版本

4.10.1 导引

SIMULINK具有帮助用户管理模型版本信息的能力。当用户编辑一个模型时，SIMULINK会自动生成该模型的版本控制信息，包括模型的版本号、作者姓名、模型最近的一次更新以及以往对模型所作的修改等。SIMULINK在保存模型的同时自动保存模型的版本信息。

SIMULINK的模型参数对话框可以让用户修改储存在模型当中的版本信息，设置其中的某些选项。Model Info模块则可以用来显示版本信息的内容。同时，SIMULINK的版本控制指令可以让用户在命令窗口或M文件中访问版本信息。

4.10.2 指定当前的用户

当用户创建或修改某个模型时，SIMULINK会自动将用户的名字登记到模型的版本信息中去。SIMULINK假定用户的名字是由USER，USERNAME，LOGIN，或LOGNAME等其中的一个变量指定。如果系统没有定义以上任何变量，则SIMULINK不会更新模型当中的用户名。

不同的操作系统对用户名的定义方式也不同。UNIX系统定义了USER的环境变量，并且指定为用户的登录名。而Windows系统根据版本的不同，可能需要手动进行设置。MATLAB中的getenv指令可以确定系统定义了哪一个环境变量。例如，在MATLAB的命令窗口中输入：

```
>>getenv('user')
```

该指令可以确定USER 环境变量是否存在于Windows系统中。如果没有，用户必须自己设置。对于Windows 95 和98系统，可以在autoexec.bat文件中通过下面的指令进行设置用户名：

```
>>set user=yourname
```

该文件一般放在C:盘的根目录下。

对于Windows NT系统，则可以在系统属性对话框中进行设置。

4.10.3 模型属性对话框

SIMULINK中的模型属性对话框允许用户设置模型当中的版本信息及相关选项。打开模型属性对话框的方法是在SIMULINK窗口中选择“File：Model Properties”菜单，出现的模型属性对话框如图4.40所示。

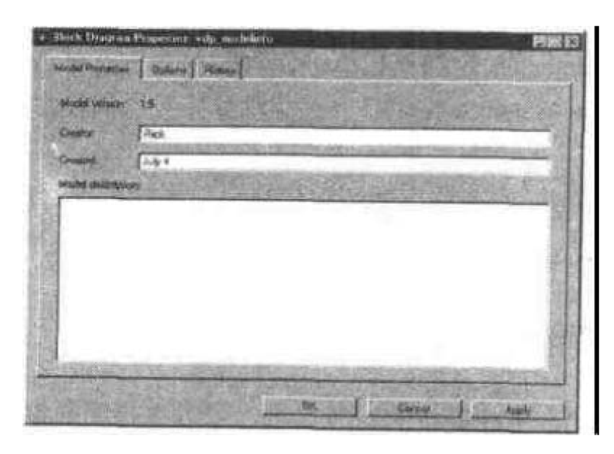


图 4.40 模型属性对话框

各部分的含义如下：

（1）Model Properties 页

Creator：模型的作者，缺省时由 SIMULINK 在模型创建时设置成环境变量中 USER 的值，我们也可以修改成其他的名字。

Created.：模型创建的时间和日期。

description：对模型的简单描述。

（2）Options Pane 页

Configuration manager：指定用来管理该模型的外部配置管理器。在下拉菜单中选择其中一项，可以让用户在 Model Info 模块中显示配置管理器的内容。缺省的配置为

Default(none)：表示 Model Info 模块中不能显示配置管理器的内容。

Model version format：指定在Model Properties 页和Model Info 模块中显示版本号的格式。该参数可以设置成任何的文本字符串。也可以采用%<AutoIncrement:#>的形式，SIMULINK 将会自动用数字替换其中的标志。例如，可能将“1.%<AutoIncrement:2>”显示为“1.2”。

每当用户保存模型一次，SIMULINK 就会将其中#对应的数字加 1，例如，将模型保存后，将“1.%<1.%<AutoIncrement:2>”变成“1.%<1.%<AutoIncrement:3>”。这时，该模型的版本号为 1.3。

“Modified by” format：指定在 History 页、历史日志和 Model Info 模块中显示 Last modified by 值的格式。该参数中可以包含%<Auto>的标签，SIMULINK 将会自动用环境中的 USER 变量的值替换它。

“Modified date” format.：指定在 History 页、历史日志和 Model Info 模块中显示 Last modified date 值的格式。该参数中可以包含%<Auto>的标签，SIMULINK 将会自动用当前的时间和日期替换它。

（3）History 页

使用户浏览并编辑模型修改的日志信息。

Last modified by：最后一次修改模型的作者姓名，SIMULINK 在用户保存模型时将它设置成环境中的 USER 变量的值。该项用户不能修改。

Last modified date：最后一次修改模型的日期和时间，SIMULINK 在用户保存模型时将它设置成当前的日期和时间。该项用户不能修改。

Modified history update：指定当该模型被保存时是否进行修改日志的更新。例如，如果选择 Prompt for Comments When Save，则当保存已经修改过的模型时，SIMULINK 将会更新模型中的修改日志信息。

Modified history：模型的修改情况的历史信息。每当模型保存时由 SIMULINK 自动进行设置。我们也可以通过单击“Edit”按钮来手工编辑。

4.10.4 模型修改日志的创建与编辑

如果希望 SIMULINK 生成模型的修改日志，可以在模型属性对话框的 History 页中将 Modified history update 选项设置成 select Prompt for Comments When Save。这样，每当用户修改模型后进行保存时，SIMULINK 将弹出对话框显示这次修改的信息，用户可以按下“Save”键将这些内容添加到型修改日志中去。

如果用户对模型的修改日志内容不满意，可以单击模型属性对话框的 History 页中的“Edit”按钮。SIMULINK 将弹出修改日志对话框，如图 4.41 所示，用户可以在其中修改日志的文本，并按下“Apply”或“OK”按钮进行确认。

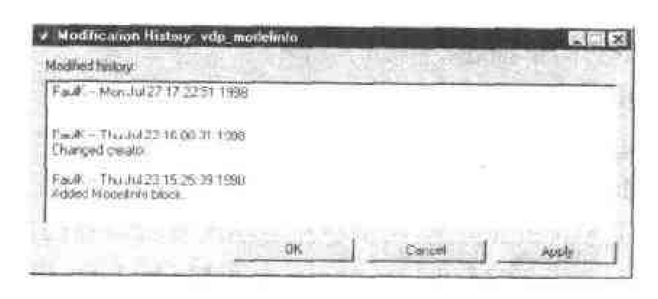


图 4.41　编辑模型的修改日志

第 5 章　SIMULINK 仿真模块库

从前面的介绍读者可以知道，创建 SIMULINK 仿真模型的基本步骤就是根据需要从模块库中选用合适的模块并以一定的方式搭构成所需要的模型。SIMULINK 为了让用户能够迅速创建所需要的系统模型，提供了用以实现各种基本功能的大量标准模块，并根据其功能的不同将它们归类在不同的模块库中。读者要掌握 SIMULINK 软件包的用法，就不能不首先熟悉这些标准模块的含义和各自的使用特点。

SIMULINK 中含有的模块非常多，并且还有很多的扩展模块，按照其功能，SIMULINK 将它们分成 10 个大类，即 Sources、Sinks、Discrete、Continuous、Nonlinear、Math、Functions & Tables、Signal &Systems、Blocksets and Toolboxes 和 Demos 等。

SIMULINK 除了提供标准模块库外，还提供了大量的适用于各种专业领域的应用模块集（Blockset）、工具箱（Toolbox）和其他辅助工具，如 Stateflow、Real-time Workshop 等。这些内容涉及到各种专业知识或是为了满足用户特殊需要。

5.1　SIMULINK 库

5.1.1　SIMULINK 库简介

SIMULINK 库是 SIMULINK 仿真环境的基本组成部分，其中收集的模块库大部分是建模时经常用到的，除非专业性很强的系统，对于一般的系统建模仿真，SIMULINK 库就完全可以胜任了。其中，SIMULINK 库又可以分为标准库与扩展库。SIMULINK 的标准库是学习 SIMULINK 建模与仿真的基础，初学者可以从标准库开始由浅入深的学习和了解 SIMULINK 中丰富的模块资源。

5.1.2　标准 SIMULINK 模块库

SIMULINK 的标准模块库中包含的模块库有：

Continuous（连续系统模块库）：主要完成线性连续系统的一些基本功能。

Discrete（离散系统模块库）：要完成线性离散系统的一些基本功能。

Function & Tables（函数与表模块库）：主要包括通用函数模块和查询表模块。

Maths（数学运算模块库）：主要完成基本的数学运算。

Nonlniear（非线性系统模块库）：主要包括处理非线性环节的模块；

Sinks（输出接受模块库）：主要包括常用的输出模块。

Signals & Systems（信号与系统模块库）：包含与模块信号和子系统相关的模块，

Source（输入源模块库）：主要包括信号发生器等信号输入模块。

各种模块库的图标如图 5.1 所示。

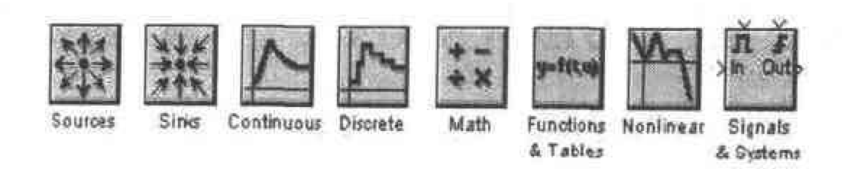


图 5.1　SIMULINK 标准库中的模块库

5.1.3　SIMULINK 扩展库（Simulink Extras）

MULINK 扩展库包含的模块库有：Additional Discrete（扩展离散库），Additional Linear（扩展线性库），Additional Sinks（扩展接受库），Flip Flops（触发模块库），Linearization（线性化库），Transformations（转换库），相应的图标如图 5.2 所示。

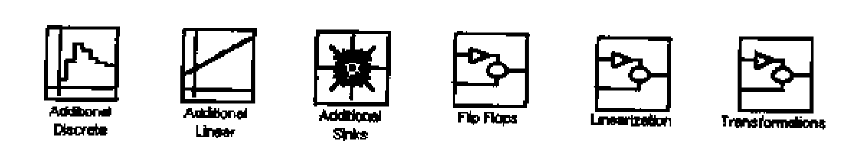


图 5.2　SIMULINK 扩展库中的模块库

（1）扩展离散库包含的模块如图 5.3 所示。包括两个离散传递函数模块（具有初始状态，初始输出各一个），两个离散零极点模块（具有初始状态、初始值各一个）。

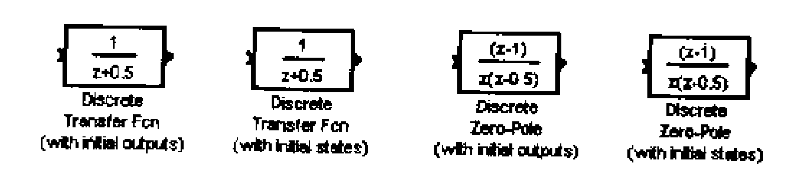


图 5.3　扩展离散库中的模块

（2）扩展线性库包含的模块如图 5.4 所示。包括两个传递函数模块（具有初始状态、初始输出各一），两个零极点模块（具有初始状态、初始输出各一），一个状态空间模块（具有初始输出），两个 PID 控制器模块（其中一个具有近似导数）。

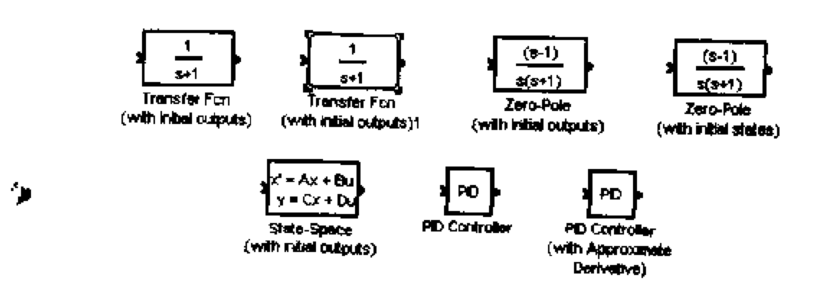


图 5.4　扩展线型库中的模块

（3）扩展接受库包含的模块如图 5.5 所示。包括：功率谱密度模块、平均功率谱密度模块、谱分析模块、平均谱分析模块、互相关器模块和自相关器模块。

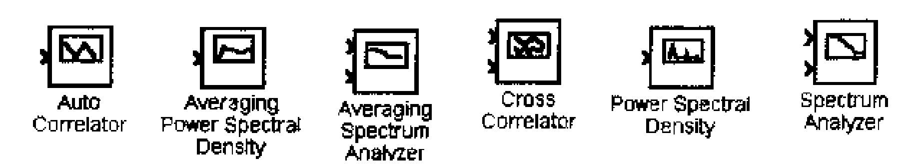


图 5.5　扩展接受库包含的模块

（4）触发模块库包含的模块如图 5.6 所示。包括：时钟模块、D 锁存器模块、S-R 触发器模块、D 触发器模块和 J-K 触发器模块（负沿触发）。

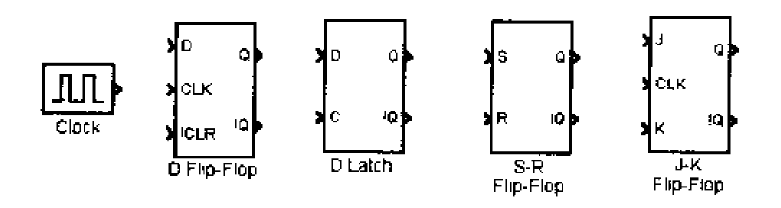


图 5.6　触发模块库中的模块

（5） 线性化库中包含的模块如图 5.7 所示。包括线性化转化导数、转化传递延迟两个模块。

（6）转化库中包含的模块如图 5.8 所示。包括极坐标与笛卡尔坐标互换两个模块、华氏温度与摄氏温度相互转换两个模块，球坐标与笛卡尔坐标相互转化两个模块，度与弧度相互转换两个模块。

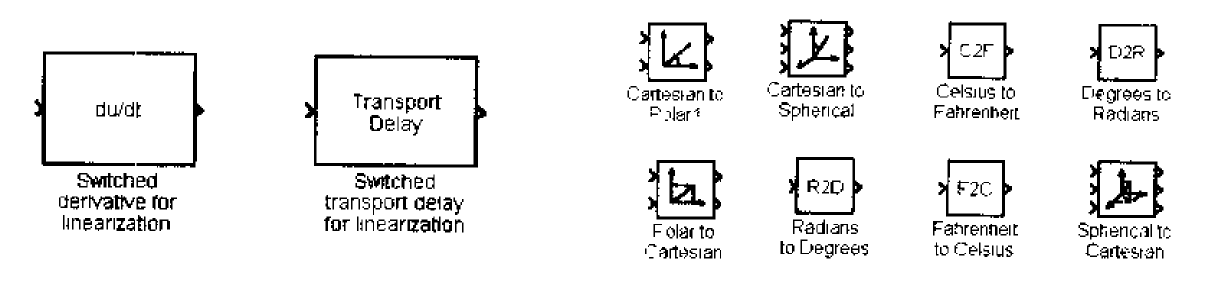


图 5.7　线性化库中的模块　　图 5.8　转化库中的模块

5.2　SIMULINK 模块集

5.2.1　通信模块集（Communications Blockset）

通信模块集如图 5.9 所示。

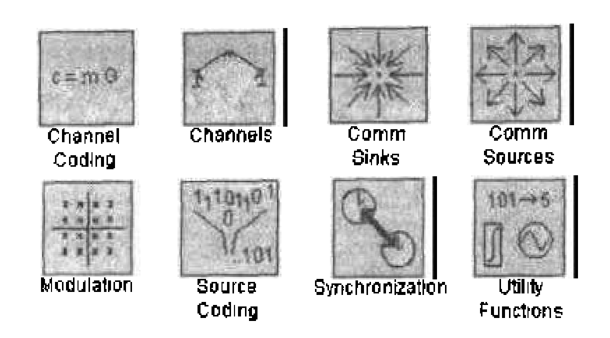


图 5.9 通信模块集中的模块库

其中包含 8 个模块库：

（1）信道编码库（Channel Coding），包括模块编码库和卷积编码库。

模块编码库中又包含各种编码和解码成对模块以及相应的演示模块。

- 线性编码模块组，包括二进制向量线性编码、解码和演示三个模块，二进制序列线性编码、解码和演示三个模块。
- 循环编码模块组，包括二进制向量循环编码、解码和演示三个模块，二进制序列循环编码、解码和演示三个模块。
- Hamming 编码模块组：包括二进制向量 Hamming 编码、解码和演示三个模块，二进制序列 Hamming 编码、解码和演示三个模块。
- BCH 编码模块组，包括二进制向量 BCH 编码、解码和演示三个模块，二进制序列 BCH 编码、解码和演示三个模块。
- Reed-Solomon 编码模块组，包括整数向量 RS 编码、解码和演示三个模块，二进制向量 RS 编码、解码和演示三个模块，整数序列 RS 编码、解码和演示三个模块，二进制序列 RS 编码、解码和演示三个模块。

卷积编码库中包含：卷积编码、Viterbi 编码和演示三个模块。

（2）信道库（Channels），包括：

- 加零均值 Gauss 白噪声（AWGN）信道模块和四个演示模块。
- 加二进制误差信道模块及演示模块。
- 有限二进制误差模块及演示模块。
- 定参数 Rayleigh 衰减信道模块，变参数 Rayleigh 衰减信道模块及演示模块。
- 定参数加 Rician 噪声信道模块，变参数加 Rician 噪声信道模块及两个演示模块。

（3）通信接受库（Comm Sinks）

- 触发写文件模块及触发文件 I/O 演示模块。
- 触发眼孔图样/散布图模块及演示模块。
- 采样时间眼孔图样/散布图模块及演示模块。
- 误差率计算模块及演示模块。

（4）通信源库（Comm Sources）

- 触发文件读入模块及触发文件 I/O 演示模块。
- 采样读工作空间变量模块，具有同步脉冲的采样读工作空间变量模块。
- 具有采样率的向量脉冲模块。

- 伪随机序列发生器模块及演示模块。
- 均匀分布的噪声发生器模块及演示模块。
- Gauss 分布噪声发生器模块及演示模块。
- 随机整数发生器模块及均匀分布整数演示模块。
- Poisson 分布随机整数发生器模块及演示模块
- 二进制向量发生器模块及演示模块。
- Bernoulli 分布随机数发生器模块及演示模块。
- Rayleigh 分布随机噪声发生器模块及演示模块。
- Rician 分布噪声发生器模块及演示模块。

（5）调制库（Modulation），包含数字基带调整模块库，数字通带调制模块库，模拟基带调制模块库和模拟通带调制模块库。

数字基带调制模块库包括：

- 基带 MASK（Multiple Amplitude Shift Keying）调制、解调及演示三个模块。
- 基带 S-QASK（Quadrature Amplitude Shift Keying）调制、解调和演示模块。
- 基带 A-QASK 调制、解调和演示模块。
- 基带 MFSK（Multiple Frequence Shift Keying）调制模块，基带相干 MFSK 解调模块，基带非相干 MFSK 解调模块和它们的演示模块。
- 基带 MPSK（Multiple Phase Shift Keying）调制、解调和演示模块。

数字基带通带模块库包括：

- 通带 MASK 调制、解调及演示三个模块。
- 通带 S-QASK 调制、解调和演示模块。
- 通带 A-QASK 调制、解调和演示模块。
- 通带 MFSK 调制模块，通带相干 MFSK 解调模块，通带非相干 MFSK 解调模块和它们的演示模块。
- 通带 MPSK（Multiple Phase Shift Keying）调制、解调和演示模块。
- 通带 DPSK（Differential Phase Shift Keying）调制、解调模块。
- 通带 MSK（Minimun Phase Shift Keying）调制、解调模块。
- 通带 OQPSK（Offset Quadrature Phase Shift Keying）调制、解调模块。

模拟基带调制模块库包括：

- 基带 DSB-SC（Double Side Band Shift Control）调幅、解调和演示三个模块。
- 基带 QAM（Quadrature Amplitude Modulation）调制、解调和演示模块。
- 基带 FM（Frequency Modulation）调频、解调和演示模块。
- 基带 PM（Phase Modulation）相位调制、解调和演示模块。
- 基带 SSB-AM（Single Side Band Amplitude Modulation）单边带调幅、解调及演示三个模块。
- 具有传输载波的基带 AM、解调和演示三个模块。

模拟通带调制模块库包括：

- 通带 DSB-SC 调幅、解调和演示三个模块。
- 通带 QAM 调制、解调和演示模块。
- 通带 FM 调频、解调和演示模块。

- 通带 PM 相位调制、解调和演示模块。
- 通带 SSB-AM 单边带调幅、解调及演示三个模块。
- 具有传输载波的通带 AM、解调和演示三个模块。

（6）源编码器（Source Coding）包括：

- 标量量化编码、解码和演示三个模块。
- 激活量化编码和演示两个模块。
- DPCM（Differential Pulse Code Modulation）编码、解码及演示三个模块。
- 规则压缩、解压模块。
- A 规则压缩、解压模块。

（7）同步库（Synchronization）包括：

- PLL（Phase Locked Loop）模块，基带 PLL 模型模块及演示模块。
- 进料泵 PLL 模块。
- 线性化基带 PLL 模块。

（8）实用函数库（Utility Functions）包括：

- 离散时间积分器模块。
- 模积分器模块。
- 离散 VCO（Voltage Controlled Oscillator）模块。
- VCO 模块。
- 可复位数字计数器模块。
- 错误计数器模块。
- 数据绘图器及演示模块。
- 二进制微分编码器和解码器模块。
- 窗口积分器模块。
- 包络检测器模块。
- 十进制整数标量与向量互换转换器模块。
- 交错模块及演示模块。
- 预定复位积分模块。
- 信号边沿检测模块。
- 扰频器、解扰器及演示模块。
- 寄存器移位及演示模块。
- 触发缓冲器模块。
- 触发向量信号重新分布及演示两个模块。
- 向量信号重新分布及演示两个模块。

5.2.2 面板与仪表模块集（Dials & Gauges Blockset）

面板与仪表模块集采用 ActiveX 技术，将常用的仪器仪表如开关、仪表盘、指示灯等包含进 SIMULINK 仿真环境中，为用户提供了与仿真模型进行交互的接口，同时扩展了其仿真的范围。其中包含的模块库有：基于面板的仪器库（Dashboard-Based Instrumentation）和基于模型的仪器库（Model-Based Instrumentation）。

5.2.3　数字信号处理模块（DSP Blockset）

数字处理模块集如图 5.10 所示，其中包含的模块库有：

（1）DSP 接受库（DSP Sinks）。

（2）DSP 源库（DSP Sources）。

（3）估计库（Estimation），包括有参数估计模块和功率谱估计模块库。

参数估计模块库：

- Yule-Walker AR 模块
- Burg AR　估计模块
- 协方差 AR 估计模块
- 改进的协方差 AR 估计模块

功率谱估计模块库：

- 短时 FFT 模块
- 幅值 FFT 模块
- Yule-Waker AR　谱估计模块
- Burg AR 谱估计模块
- 协方差 AR 谱估计模块
- 改进的协方差 AR 谱估计模块

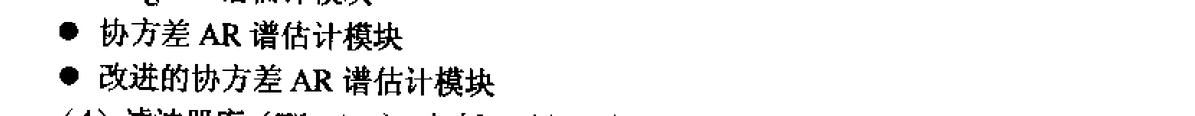














图 5.10　数字处理模块集中的模块库

（4）滤波器库（Filtering），包括：滤波器模块库、滤波器实现模块库，自适应滤波器模块库，多级滤波器模块库等。

（5）通用 DSP 库（General DSP），包含信号操作、信号变换、信号缓冲、开关与计数器四个模块库。

（6）数学函数库（Math Functions），包括基本函数、向量函数、矩阵函数、线性代数和统计 5 个模块库。

5.2.4　定点模块库（Fixed-Point Blockset）

定点模块集如图 5.11 所示，

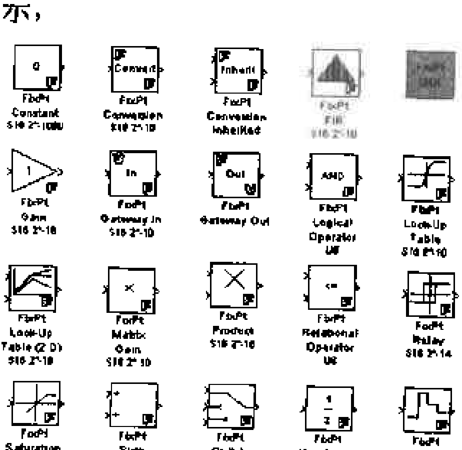
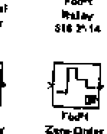
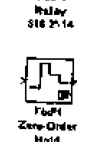




图 5.11　定点模块集中的模块库与模块

定点模块集包含的模块库与模块有：

- 滤波器与系统模块库（Filters and systems Examples）
- 定点参数模块（FixPt Constant）
- 定点转换模块（FixPt Conversion）
- 定点遗传转换模块（FixPt Conversion Inherited）
- 定点 FIR 模块（FixPt FIR）
- 定点 GUI 模块（FixPt GUI）
- 定点增益模块（FixPt Gain）
- 定点门输入模块（FixPt Gateway In）
- 定点门输出模块（FixPt Gateway Out）
- 定点逻辑运算模块（FixPt Logical Opertator）
- 定点查询表模块（Look-Up Table）
- 二维定点查询表模块（FixPt Look-Up Table 2-D）
- 定点矩阵增益模块（FixPt Matrix Gain）
- 定点乘积模块（Fixed Product）
- 定点关系运算模块（FixPt Relation Opertator）
- 定点延迟模块（FixPt Relay）
- 定点饱和模块（FixPt Saturation）
- 定点求和模块（FixPt Sum）
- 定点开关模块（FixPt Switch）
- 定点单位延迟模块（FixPt Unit Delay）
- 定点零阶保持模块（FixPt Zero-Order Hold）

5.2.5　非线性控制系统设计模块集（NCD Blockset）

NCD 模块集如图 5.12 所示，其中包含的模块库有：

（1）RMS 模块库，包括连续、离散 RMS 模块和它们的演示模块。

（2）NCD 输出端口模块库（NCD Outport）。

（3）NCD 演示模块库（NCD Demos），包括含有四个演示模块和三个指南模块。

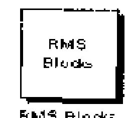
5.2.6　神经网络模块集（Neural Network Blockset）

神经网络模块集如图 5.13 所示，包含的模块库有：

（1）传递函数模块库（Transfer Functions）。

（2）权函数模块库（Weight Functions）。

（3）网络输入函数库（Net Input Functions）。



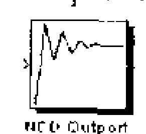
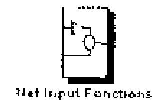


图 5.12　NCD 模块集中的模块库与模块



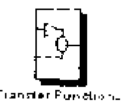
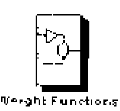



图 5.13　神经网络模块集中的模块库

5.2.7　MPC 模块集（MPC Blockset）

包含 nlcmpc 模块和 nlmpcsim 模块，如图 5.14 所示。

5.2.8　电力系统模块集（Power System Blockset）

电力系统模块集如图 5.15 所示，其中包含的模块库有

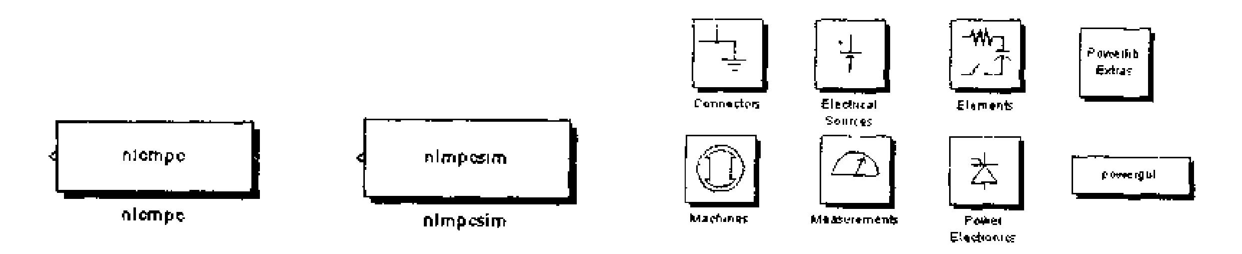


图 5.14　MPC 模块集　　　　图 5.15　电力系统模块集包含的模块库

（1）电源模块库

- 直流电压源模块（DC Voltage Source）
- 交流电压源模块（AC Voltage Source）
- 交流电流源模块（AC Current Source）
- 可控电压源模块（Controlled Voltage Source）
- 可控电流源模块（Controlled Current Source）

（2）元件模块库

- 串联 RLC 支路模块（Series RLC Branch）
- 串联 RLC 负载模块（Series RLC Load）
- 并联 RLC 支路模块（Parallel RLC Branch）
- 并联 RLC 负载模块（Parallel RLC Load）
- 线性变压器模块（Linear Transformer）
- 饱和变压器模块（Saturable Transformer）
- 互感器模块（Mutual Inductance）
- 电涌放电器模块（Surge Arrester）
- 分布参数线路模块（Distributed Parameters Line）
- 断路器模块（Bleaker）
- π 截面导线模块（PI Section Line）

（3）电源电子元件模块库

- 理想开关模块（Ideal Switch）
- 金属氧化物半导体模块（Diode）
- 场效应晶体管模块（Mosfet）
- 门电路模块（Thyristor）
- 二极管模块（Gto）

- 可控硅模块（Detailed Thyrisor）

（4）仪表模块库

- 电压测量模块（Voltage Measurement）
- 电流测量模块（Current Measurement）

（5）连接器模块库

- 接地模块
- 局部接地模块
- T 型和 L 型接地模块
- 多进多出连接器
- 多进多出薄连接器

（6）电机模块库

- 简单的同步电机模块（3 个）
- 恒磁同步电机模块（2 个）
- 异步电机模块（3 个）
- 涡轮与调节器模块（2 个）
- 同步电机模块（4 个）

（7）附加电源模块库，包括测量模块库、控制模块集和附加电机模块库等。

5.3　其他辅助工具

5.3.1　实时窗口目标库（Real-Time Windows Target）

实时窗口目标库如图 5.16 所示。其中包括适配器模块（Adapter）、实时输入单元模块（RT In）和实时输出单元模块（RT Out）。

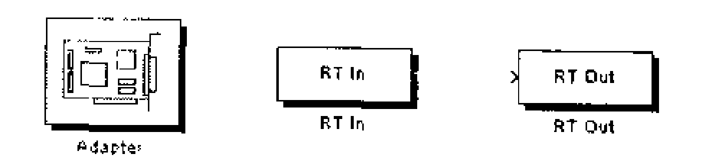


图 5.16　实时窗口目标库

5.3.2　实时工作室（Real-Time Workshop）

实时工作室如图 5.17 所示，包含的模块库有：

（1）自定义代码库(Custom Code)

（2）DOS 设备驱动器库（Dos Device Drivers）

（3）中断模块库（Interrupt Templates）

（4）S 函数对象库（S-Function Target）

（5）Vx 工作支持库（VxWorks Support）

以上模块库中又可能包含子模块库。

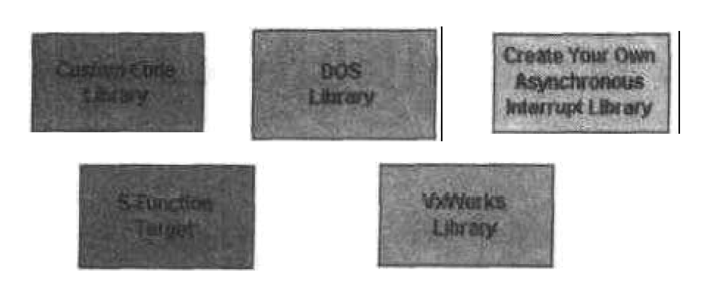


图 5.17　实时工作室包含的模块库

5.3.3　状态流模块库（Stateflow）

状态流模块库包含状态例程图模块（Chart）和演示库。其中双击流程图图标，可以打开状态流程图窗口（如图 5.18 所示）。

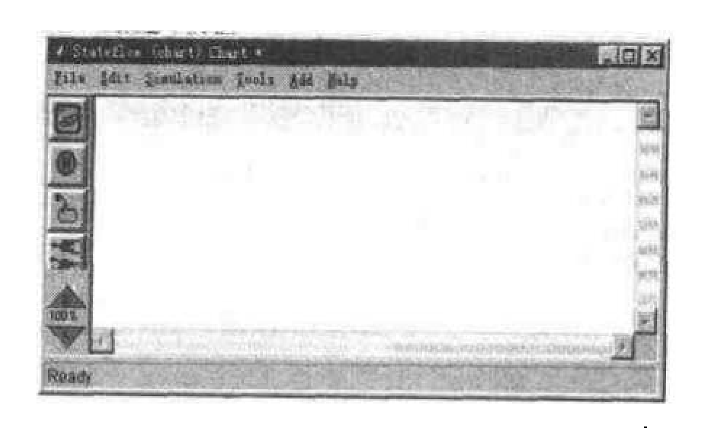


图 5.18　状态流程图窗口

第6章　SIMULINK 模块库索引

SIMULINK 标准模块库中包含了用户最常用的模块，熟悉这些模块的属性和用法对仿真模型的设计和创建方框图将有很大的帮助。本章将详细介绍这些 SIMULINK 标准模块的用法，包括图标、属性对话框、参数设置以及特性等等。

6.1　Source 库

6.1.1　Band-Limit White Noise 模块

Band-Limit White Noise 的属性对话框如图 6.1 所示。

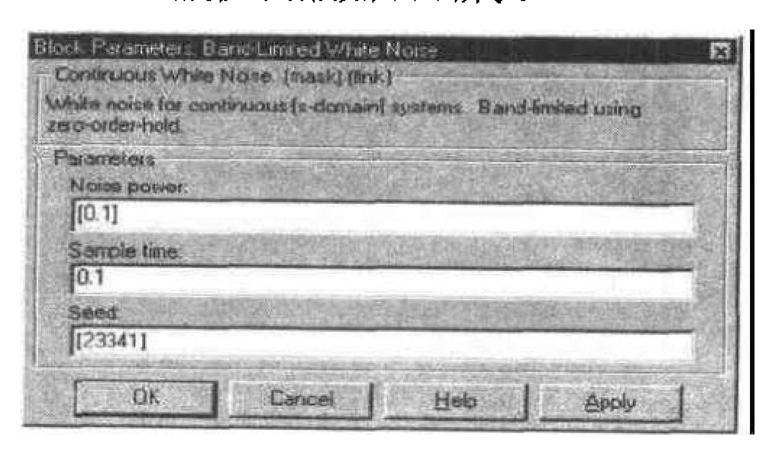


图 6.1　Band-Limit White Noise 模块的属性对话框

说明：产生正态分布的随机数，适用于连续系统或者混合系统。它与 Random Number 模块的主要区别在于前者以一给定的采样频率产生输出，而该模块的采样频率与噪声的相关时间有关。

理论上，连续白噪声的相关时间为 0，功率谱密度（PSD）是平坦的，协方差无限大。但实际上系统的干扰信号不是白噪声，当干扰噪声的相关时间相对于系统的带宽非常小时，可以近似采用白噪声。

在 SIMULINK 中，可以用相关时间比系统的最短时间常数小得多的随机序列来模拟实际噪声。该模块就是产生这样的随机序列，其噪声的相关时间是模块的采样速率。要想精确的进行仿真，就必须使用比系统最快的动态分量还要小得多的相关时间。按（6.1）式给定相关时间会得到比较好的结果。

$$t_c = \frac{2\pi}{100 f_{max}} \tag{6.1}$$

数据类型　　支持双精度实数类型的信号。
参数　　Noise power：噪声的功率谱（PSD）的大小，缺省值为 0.1。
Sample time：噪声的相关时间，缺省值为 0.1。
Seed：产生随机数的种子，缺省值为 23341。
特点　　采样时间　　离散
标量扩展　　参数或输出扩展
向量化　　可以
零点穿越　　无

6.1.2　Chirp Signal 模块

Chirp Signal 模块的属性对话框如图 6.2 所示。

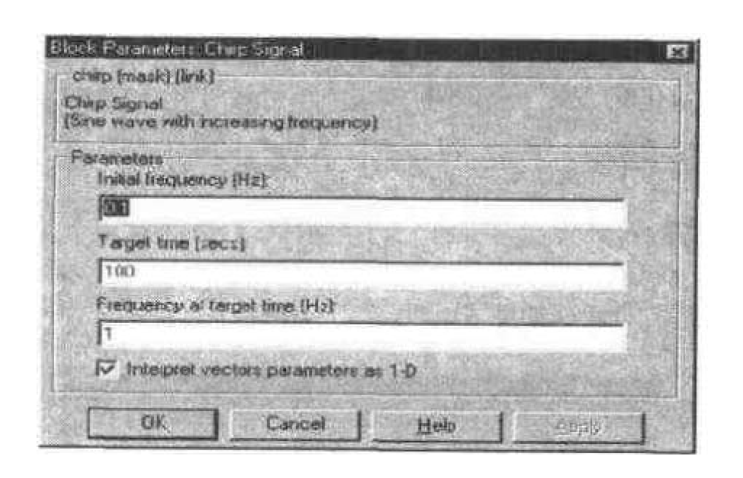


图 6.2　Chirp Signal 模块的属性对话框

说明：产生一个频率随时间线性递增的正弦波信号，可以用该模块进行非线性系统的频谱分析。模块的输出为标量或向量。

数据类型支持双精度实数类型的信号。
参数　　Initial frequency：信号的初始频率，可以为向量，缺省值为 0.1。
Target time：频率达到目标频率参数的时间值，之后保持输出频率不变。
Frequency at target time：目标时间的频率，可以为标量或向量。
Interpret vector parameters as 1-D：如果在选中状态，则模块参数的行或列值将转换成向量进行输出。
特点　　采样时间　　连续
标量扩展　　参数扩展
向量化　　可以
零点穿越　　无

6.1.3　Clock 模块

Clock 模块的属性对话框如图 6.3 所示。

说明：该模块输出每一步仿真的当前仿真时刻。本模块对于其他需要仿真时间的模块来说是有用的。

如果要得到离散系统的当前仿真时间，可以采用 Digital Clock 模块。

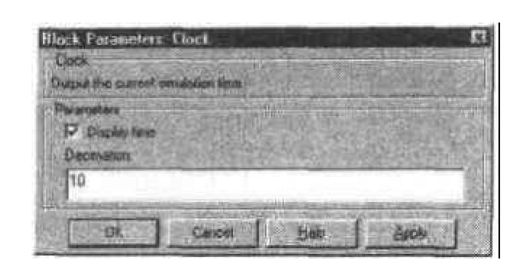


图 6.3　Clock 模块的属性对话框

数据类型　输出双精度型的实数信号。

参数　Display time ：是否在模块图标中显示当前的仿真时间。

Decimation：可以是任何的非负整数，它设置了时钟信号更新的频率，单位是 1/1000 秒。例如如果设为 1000，则该模块每隔 1 秒输出时间信号。注意，如果该值不为零，SIMULINK 将强制采用固定步长的求解算法，以满足时钟输出的需要。

特点

采样时间	连续
标量扩展	不适用
向量化	不能
零点穿越	无

6.1.4　Constant 模块

Constant 模块的属性对话框如图 6.4 所示。

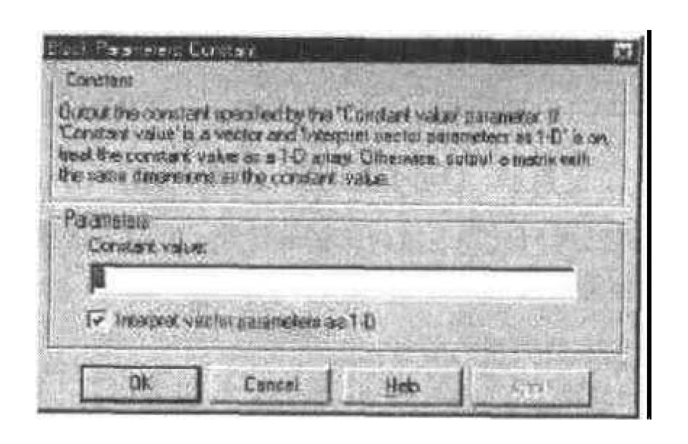


图 6.4　Constant 的属性对话框

说明：该模块产生一个不依赖于时间的实数或复数的常数值，一般作为定常输入信号。模块输出可以为标量、向量或矩阵，具体取决于模块参数和 Interpret vector parameters as 1-D 参数的设置．在 Interpret vector parameters as 1-D 被选中的情况下，如果模块参数值是行或列向量，则输出信号为一维向量，否则输出为矩阵。

数据类型　支持实数或复数，输出数据类型与模块参数相同。

参数　Constant value：模块的常值输出，可以为向量。

Interpret vector parameters as 1-D：如果在选中状态，则模块参数的行或列向量将转换成向量进行输出。

特点	采样时间	固定值
	标量扩展	不适用
	向量化	可以
	直通特性	无

6.1.5　Digital Clock 模块

Digital Clock 模块的属性对话框如图 6.5 所示。

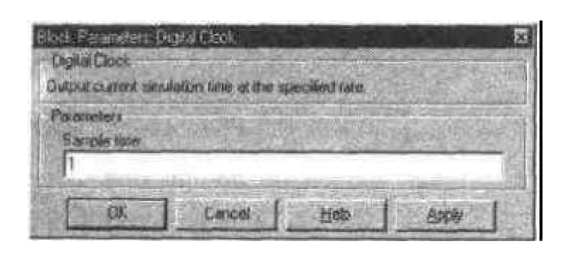


图 6.5　Digital Clock 模块的属性对话框

说明：该模块输出指定采样时间间隔的仿真时间，其他时间模块输出保持前一时刻的值不变。适用于离散系统。

数据类型　支持双精度实数类型的信号。

参数　Sample time ：采样的时间间隔，缺省值为 1 秒。

特点	采样时间	离散
	标量扩展	无
	向量化	无
	零点穿越	无

6.1.6　Discrete Pulse Generator 模块

Discrete Pulse Generator 模块的属性对话框如图 6.6 所示。

图 6.6　Discrete Pulse Generator 模块的属性对话框

说明：以一定的时间间隔产生一系列的脉冲信号。脉冲的宽度是脉冲为高电平时的采样周期的个数。周期是脉冲为一个高电平和一个低电平时采样周期的个数。相位延迟时在脉冲开始前采样的周期数目。相位延迟可以是正数也可以为负数但不大于一个周期。采样时间必

须大于 0。

在离散和混合系统中可以使用该模块，但如果要产生一个连续信号，应该使用 Pulse Generator 模块。

数据类型　支持双精度实数类型的信号。

参数　Amplitude ：输出脉冲信号的幅值，缺省值为 1。

Period：以采样数为脉冲周期，缺省值为 2。

Pulse width：脉冲为高电平时的采样周期数，缺省值为 1。

Phase delay：脉冲发生前的延迟采样周期数，缺省值为 0。

Sample time：采样周期的大小，缺省值为 1 秒。

Interpret vector parameters as 1-D：如果在选中状态，则模块参数的行或列向量将转换成向量进行输出。

特点

采样时间	离散
标量扩展	参数扩展
向量化	可以
零点穿越	无

6.1.7 From Workspace 模块

From Workspace 模块的属性对话框如图 6.7 所示。

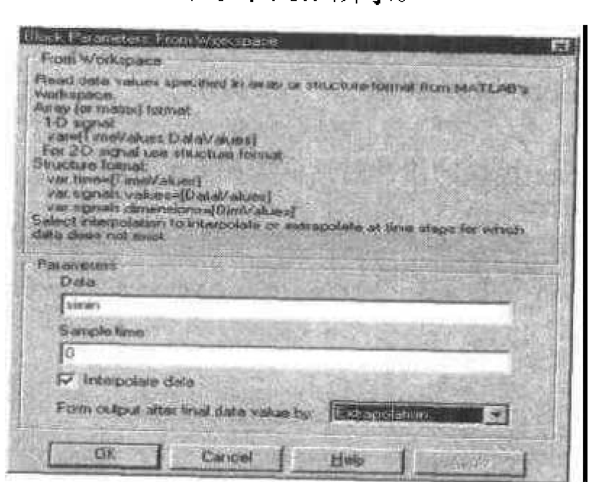


图 6.7　From Workspace 模块的属性对话框

说明：该模块从 MATLAB 的基本工作空间中读取数据。模块中的 Data 参数指定了读取数据的变量名，该变量名可以在图标中显示出来。读取的数据放在一个二维数组（矩阵）或某个结构中，其中包含仿真时间和相应的数据。

模块的 Interpolate data 参数决定了输出数据的时间间隔。如果该参数被选中，则位于两个仿真步之间的数据通过插值得到，否则，取其中距离最近的数据。

模块的 Form output after final data value by 参数决定了当从数据文件中读取完最后一个数据时该模块的输出值。

如果变量中包含相同的时间量，则SIMULINK取最后的时间量所对应的数据。如：

time:　0 1 2 2

signal:　2 3 4 5

在时间为 2 时，模块输出为 5。

数据类型　可以接受任何类型的实数或复数信号。双精度型的实数信号可以是结构型也可以是矩阵。但任何其他非双精度型的实数或复数信号都必须采用结构格式。

参数　Data：包含仿真时间向量和数据向量的矩阵或结构名。例如，如果相应的时间列向量为 T，对应的数据向量为 U，则该参数为[T，　U]。也可以直接输入工作空间中的结构或矩阵名。

Sample time：从工作空间中读取数据的采样周期。

Interpolate data：如果为选中状态，则 SIMULINK 通过线性插值得到两步仿真之间的值，否则，直接取距离最近的数据。

Form output after final data value by：确定该模块在读取完最后时刻的数据后，模块的输出值。

特点

采样时间	从驱动模块继承
标量扩展	无
向量化	有
零点穿越	无

6.1.8　From File 模块

From File 模块的属性对话框如图 6.8 所示。

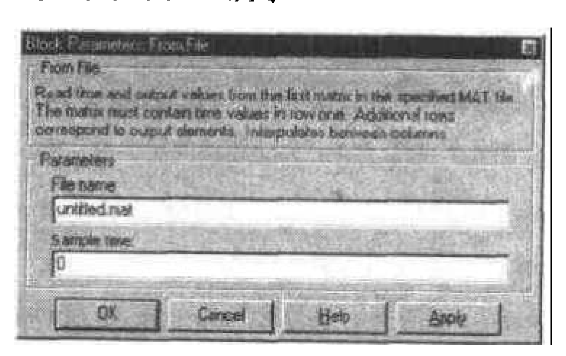


图 6.8　From File 模块的属性对话框

说明：该模块从指定的数据文件中读取数据，模块图标上会自动显示文件的路径。

数据文件至少有两行，第一行为单调递增的时间，其他行为对应的输入数据，基本形式如下：

$$\begin{bmatrix} t_1 & t_2 & \cdots & t_{final} \\ u1_1 & u1_2 & \cdots & u1_{final} \\ \cdots & & & \\ un_1 & un_2 & \cdots & un_{final} \end{bmatrix} \qquad (6.2)$$

仿真中对于数据文件没有描述对应时间的数据，采用线性插值的方法得到。使用这个模块可以设定任意的输入曲线，但是输入的数据不能太少，否则靠插值得到的数据将使仿真精度降低。

数据类型　输出双精度型的实数信号。

参数　File name ：包含载入数据的文件名。缺省时为 untitled.mat。SIMULINK 假定该文件在 MATLAB 的工作目录下（可使用 pwd 指令进行设置），否则会出现错误信息。

Sample time：从数据文件读取数据的采样周期。

特点
采样时间	从驱动模块继承
标量扩展	无
向量化	仅仅支持一维数组
零点穿越	无

6.1.9 Pulse Generator 模块

Pulse Generator 模块的属性对话框如图 6.9 所示。

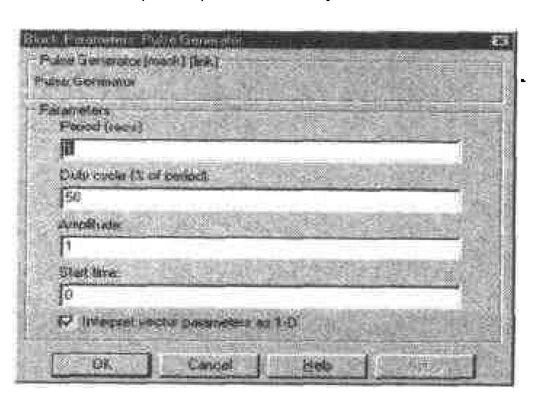


图 6.9　Pulse Generator 模块的属性对话框

说明：该模块以一定的时间间隔产生标量、向量或矩阵形式的脉冲信号。只使用于连续系统，离散系统可用 Discrete Pulse Generator 模块。

数据类型　支持任何类型的实数或复数信号，实际输出信号的类型与 Amplitude 参数类型一致。

参数　Period：脉冲周期，单位为秒，缺省值为 1。

Duty cycle：信号为高电平的时间在一个周期内的比例，缺省值为 50%。

Amplitude：脉冲的幅值，缺省值为 1。

Start time：脉冲输出开始的时间，缺省值为 0 秒。

Interpret vector parameters as 1-D：如果在选中状态，则模块参数的行或列向量将转换成向量进行输出。

特点
采样时间	从驱动模块继承
标量扩展	参数扩展
向量化	可以

零点穿越　　无

6.1.10　Ramp 模块

Ramp 模块的属性对话框如图 6.10 所示。

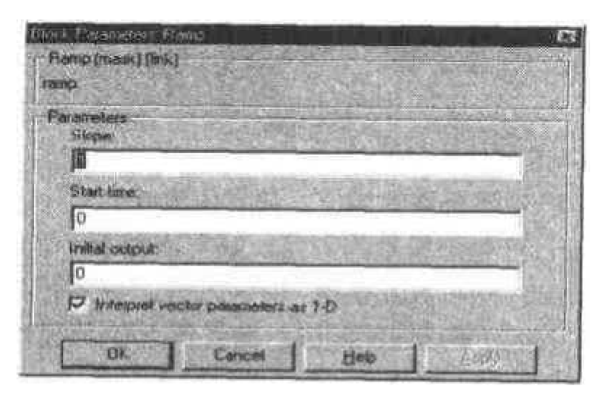


图 6.10　Ramp 模块的属性对话框

说明：该模块产生斜率不变的斜坡信号。斜率可以为负数。

数据类型	支持双精度类型的信号。	
参数	Slope：斜坡信号的斜率，缺省值为 1。	
	Start time：信号开始产生的时间，缺省值为 0 秒。	
	Initial output：信号的初值，缺省值为 0。	
	Interpret vector parameters as 1-D：同上。	
特点	采样时间	从驱动模块中继承
	标量扩展	有
	向量化	可以
	零点穿越	有

6.1.11　Random Number 模块

Random Number 模块的属性对话框如图 6.11 所示。

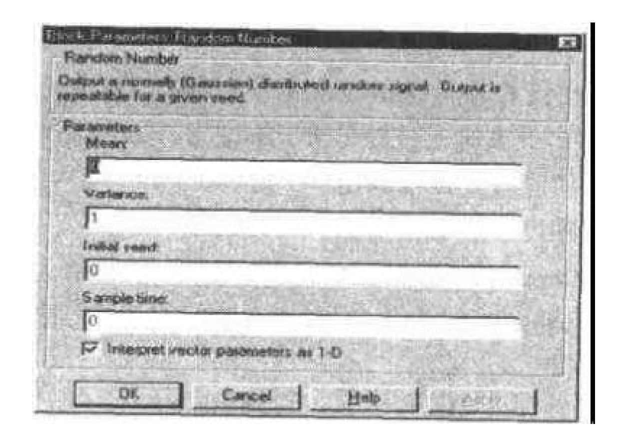


图 6.11　Random Number 模块的属性对话框

说明：该模块可以产生正态分布的随机数。每次仿真开始时随机种子设置成指定值。缺省时，产生均值为 0，方差为 1 的随机序列，产生的随机数是可重复的，可以用任何 Random Number 模块以相同的参数产生。要生成相同均值和方差的随机数向量，指定 Initial Seed 参数为向量即可。

要生成均匀分布的随机数，使用 Uniform Random Number 模块。不应该对一个随机数进行积分，因为求解器对光滑信号进行积分才有意义。此时可以改用 Band-Limited White Noise 模块替代。

数据类型　支持双精度类型信号。

参数　Mean：产生随机数的均值，缺省值为 0。

Variance：产生随机数的方差，缺省值为 1。

Initial seed：随机数发生器开始的种子，缺省值为 0。

Sample time：采样时间间隔，缺省值为 0，该模块具有连续采样时间。

Interpret vector parameters as 1-D：同上。

特点　采样时间　连续或离散

标量扩展　参数扩展

向量化　可以

零点穿越　有

6.1.12　Repeating Sequence 模块

Repeating Sequence 模块的属性对话框如图 6.12 所示。

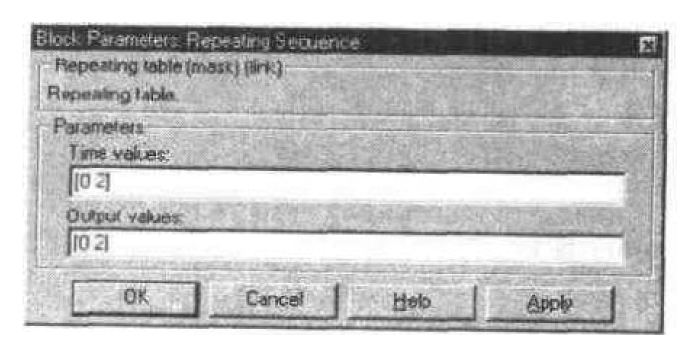


图 6.12　Repeating Sequence 模块的属性对话框

说明：该模块可以产生随着时间的推移在波形上重复的信号，波形可以任意指定，当仿真达到 Time value 向量中的最大时间值时信号开始重复。该模块是使用一维 Look-Up Table 模块实现的，在各个点之间进行了线性插值。

数据类型　输出双精度的实数信号。

参数　Time values：单调增加的时间值向量，缺省值为[0 2]。

Output values：输出值向量，每个值对应同一时间列中的时间值。

特点　采样时间　连续

标量扩展　无

向量化　无

直通特性　　无

6.1.13　Signal Generator 模块

Signal Generator 模块的属性对话框如图 6.13 所示。

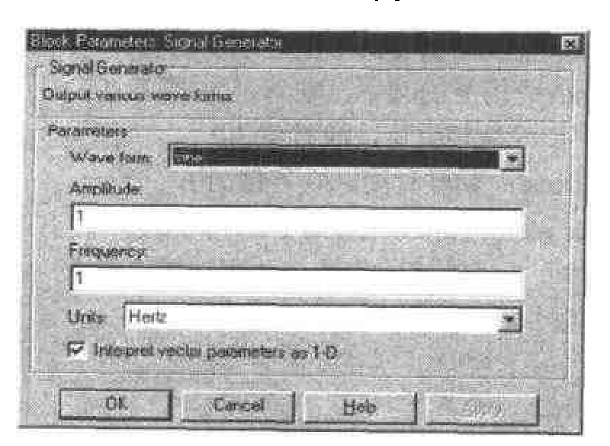


图 6.13　Signal Generator 模块的属性对话框

说明：该模块可以产生三种不同的波形：即方波、正弦波或锯齿波。频率参数的单位可以是 HZ（缺省），也可以是 rad/s。负的 Amplitude 参数可使输出波形发生 180° 偏移。可以在仿真过程中修改输出的设置，以观察不同波形下的系统响应。

模块的 Amplitude 和 Frequency 参数决定了输出的频率和幅值。参数在标量扩展后应具有同样的维数。如果 Interpret vector parameters as 1-D 选项为 off，则模块的输出与参数具有相同的维数。否则，如果模块参数是行向量或列向量形式，则输出为向量信号。

数据类型　　输出实数或复数的标量或向量形式。

参数　　Wave form ：输出波形，缺省时为正弦波。

Amplitude：输出信号的幅值，缺省为 1。

Frequency：输出信号的频率值，缺省为 1。

Units：信号频率采用的单位，是 HZ 还是 rad/sec，缺省时为 HZ。

Interpret vector parameters as 1-D：同上。

特点　　采样时间　　连续

标量扩展　　参数扩展

向量化　　可以

零点穿越　　无

6.1.14　Sine Wave 模块

Sine Wave 模块的属性对话框如图 6.14 所示。

说明：该模块提供正弦曲线，既可以是连续形式的正弦波也可以是离散形式的正弦波。输出由下式决定：

输出 = 幅值 × sin(频率×时间+相位)

Sample time 的值确定模块是工作于连续模式还是工作于离散模式。

0　缺省值，工作于连续模式；

>0　工作于离散模式；

-1　模块的工作模式与接受信号的模块相同。

数据类型	支持双精度实数类型的信号。
参数	Amplitude：输出信号的幅值，缺省值为1。 Frequency：信号的频率，缺省时为1弧度/秒。 Phase：信号的相位移，缺省值为0。 Sample time：采样时间，缺省值为0。 Interpret vector parameters as 1-D：同上。
特点	采样时间　连续、离散或从驱动模块继承 标量扩展　参数扩展 向量化　可以 零点穿越　无

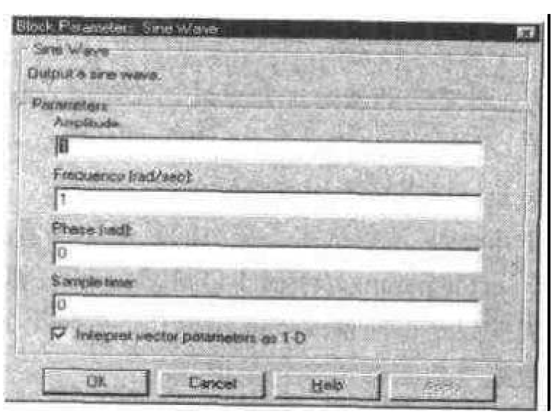


图 6.14　Sine Wave 模块的属性对话框

6.1.15　Step 模块

Step 模块的属性对话框如图 6.15 所示。

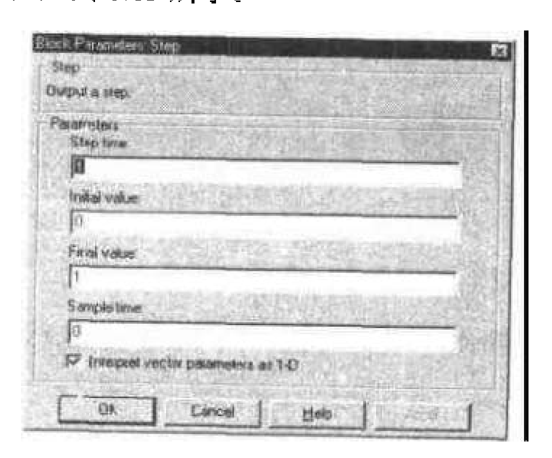


图 6.15　Step 模块的属性对话框

说明：该模块在某一指定的时刻在两值之间产生一个跳变。

如果仿真的时间小于参数 Step time 的值，该模块输出的是参数 Initial value 的值。如果仿真时间大于或等于参数 Step time 的值，输出的是参数 Final value 的值。

如果 Interpret vector parameters as 1-D 参数为 off，则模块的输出与模块参数具有相同的维数。在 Interpret vector parameters as 1-D 被选中的情况下，如果模块参数值是行或列向量，则输出信号为一维向量，否则输出与模块参数具有相同的维数。

数据类型　输出双精度型的实数信号。

参数　Step time：输出参数 Initial value 的值跳变到参数 Final value 的值的时间，单位为 s，缺省值为 1s。

Initial value：当仿真时间小于参数 Step time 的值时，该模块的输出值。缺省值为 0。

Final value：当仿真时间大于参数 Step time 的值时，该模块的输出值。缺省值为 1。

Sample time：每步的采样时间

Interpret vector parameters as 1-D：同上。

特点

采样时间	从驱动模块继承
标量扩展	参数扩展
向量化	可以
零点穿越	有

6.1.16 Uniform Random Number 模块

Uniform Random Number 模块的属性对话框如图 6.16 所示。

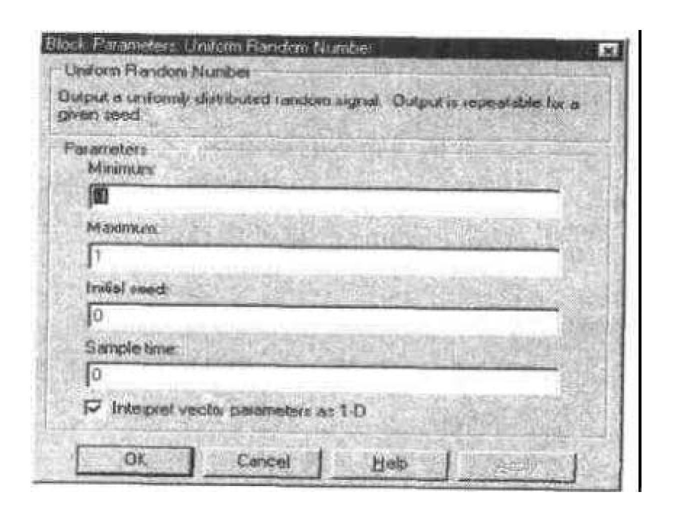


图 6.16 Uniform Random Number 模块的属性对话框

说明：该模块在指定的区间内，以指定的起始种子，生成均匀分布的随机数。在每次仿真开始时重新设置种子。生成的随机序列是可重复的并且能够由相同参数的该模块生成。要

生成随机数向量，只要指定 Initial value 参数为向量即可。

要生成正态分布的随机数，使用 Random Number 模块。

避免对随机数进行积分，因为求解器积分相对光滑信号才有意义，此时可以改用 Band-Limited White Noise 模块。

数据类型	输出双精度类型的实数信号。	
参数	Minimum：指定区间的最小值，缺省值为-1。	
	Maximum：指定区间的最大值，缺省值为 1。	
	Initial seed：随机数产生的开始种子，缺省值为 0。	
	Sample time：采样周期，缺省值为 0。	
	Interpret vector parameters as 1-D：同上。	
特点	采样时间	连续、离散或从驱动模块继承
	标量扩展	无
	向量化	可以
	零点穿越	无

6.2 Sinks 库

6.2.1 Display 模块

说明：该模块显示输入的值。可以通过选择 Format 选项来控制显示的格式：

（1）short，以短整型显示 5 位数字的定点值。

（2）long，显示缩放的 15 位数字的定点值。

（3）short_e，显示有 5 位数字的浮点值。

（4）long_e，显示有 16 位数字的浮点值。

（5）bank，显示用固定的元和分格式表示的数值。

要将该模块以浮动显示，选择 Floating display 选择框，模块的输入端口会消失，模块显示被选中的连线上的信号的值。选择 Floating display 选择框，必须关闭 SIMULINK 缓存再利用特性。

数据显示的数量和在哪些时间上显示数据由模块的参数决定。

（1）通过设定 Decimation 参数为 n，可以每 n 个采样显示一个数据，n 是降因子，缺省值为 1。这时在每一仿真步都显示数据。

（2）通过 Sample time 参数可以设置显示数据的时间间隔。这一参数对于使用变步长的求解器很有用，因为它们的时间步之间的时间间隔可能不相等。缺省值为-1，表示该模块在确定哪些点要显示时不考虑采样间隔。

如果模块输入是向量，可以改变模块图标的大小以使其显示的不仅仅是第一个元素，可以在垂直方向和水平方向上改变模块图标的大小，模块会在适当的方向增加显示区域。一个黑色的三角形表明模块没有显示出来的向量元素，如图 6.17 所示。

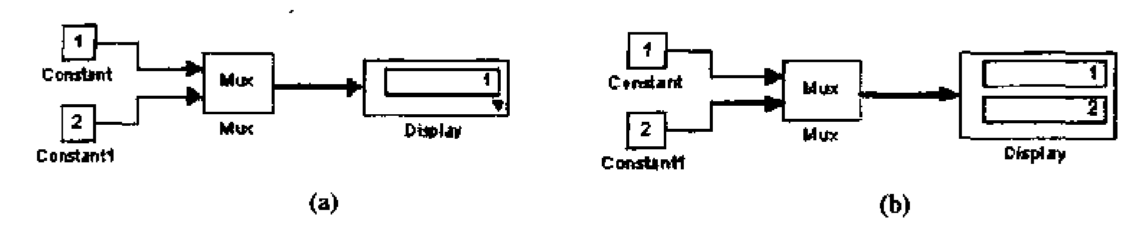


图 6.17　Display 显示向量信号的不同显示效果

数据类型	可以接受任何类型的实数或复数信号。
参数	Format：显示数据的格式，缺省值为 short。
	Decimation：显示数据的频率，缺省值为 1，显示每个输入数据。
	Floating display：是否以浮动窗口形式显示该模块。
	Sample time：显示数据的采样时间。
特点	采样时间　　从驱动模块继承
	向量化　　　可以

6.2.2　Scope 模块

Scope 模块的属性对话框如图 6.18 所示。

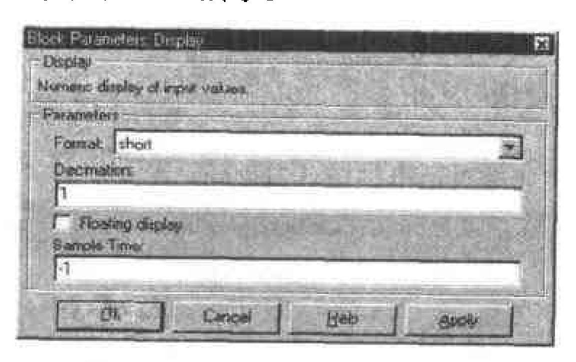


图 6.18　Display 模块的属性对话框

说明：该模块显示仿真时产生的信号曲线，横坐标为仿真时间。是最常用的模块之一。模块接受一个输入并且能够显示多个信号的图形。Scope 模块允许调整时间的大小和显示输入值的范围。可以移动 Scope 窗口、也可以改变它的大小，还可以在仿真期间改变 Scope 的参数值。

在开始仿真时，SIMULINK 并不自动打开 Scope 窗口，尽管它传送数据给有关的 Scope，因此，在仿真结束时，如果打开 Scope，Scope 窗口将会显示出来。

如果信号是连续的，Scope 模块生成由点连成的曲线，如果信号是离散的，Scope 模块生成阶梯状图形。

Scope 提供工具条按钮，可以缩放显示的数据，可以在 Scope 中显示所有的数据，可以将一个仿真中的坐标轴的设置保存给下一个仿真；可以限制显示的数据，可以保存数据到 MATLAB 工作空间中。图 6.19 显示的是 Scope 窗口。

信号选择器可以让用户在浮动 Scope 窗口中选择要显示的信号，包括没有打开的子系统

中的信号。为了使用该项功能，在浮动 Scope 窗口为打开状态下运行仿真，用鼠标右键单击 Scope 窗口，在弹出的上下文菜单中选择 “Signal Selection” 菜单项，将弹出如图 6.20 的信号选择对话框。

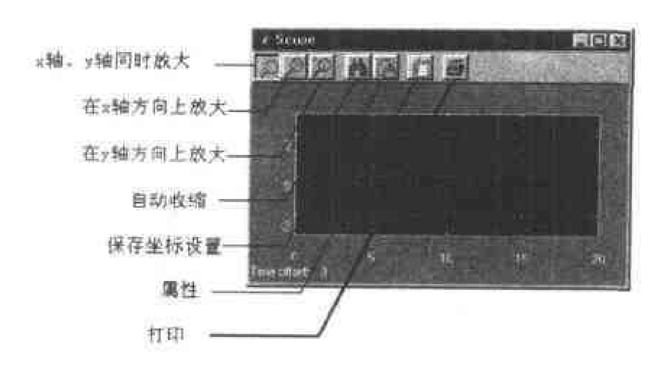


图 6.19　Scope 窗口及工具条按钮

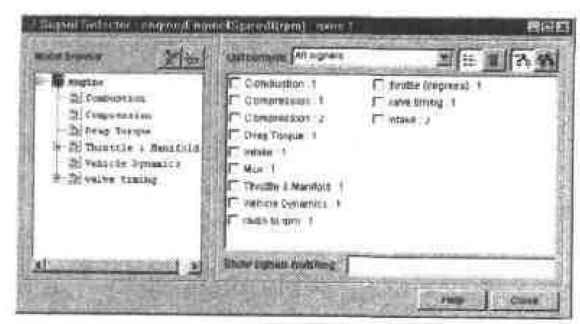


图 6.20　Scope 模块的信号选择对话框

信号选择对话框分为两个部分，左边是模型浏览器，右边显示模型中的所有信号，用户可以在其中用鼠标点选需要显示的信号。

数据类型　　可以接受实数信号，包括任何类型的同性质向量。

特点　　采样时间　　从驱动模块继承

　　状态个数　　0

6.2.3　Stop Simulation 模块

Stop Simulation 模块的属性对话框如图 6.21 所示。

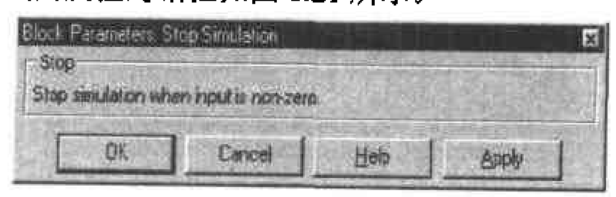


图 6.21　Stop Simulation 模块的属性对话框

说明：该模块当输入为非零值时将终止仿真过程。仿真在终止之前完成当前时间步的计算。如果该模块的输入是向量，任何非零的向量元素都会导致仿真结束。可以使用该模块与 Relational Operator 模块相连，来控制仿真的结束。

数据类型　　接受双精度或布尔类型的实数信号。

特点　　采样时间　　从驱动模块继承

　　向量化　　可以

6.2.4　To File 模块

To File 模块的属性对话框如图 6.22 所示。

说明：该模块将其输出写到 MAT 数据文件中的矩阵，它将每一时间步写成一列第一行是仿真时间；该列中剩余的行是输入的数据，输入向量中每一元素占一数据点。

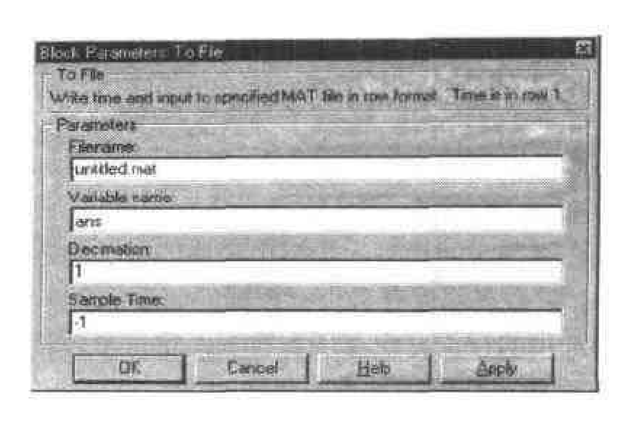


图 6.22　To File 模块的属性对话框

From File 模块能够直接使用该模块的数据，但 From File 模块得到的矩阵是该模块写入的矩阵的转置。该模块能够边仿真边写数据，在仿真结束时数据写入完成。模块的图标显示指定的输出文件的名字，如果指定文件已经存在，则在仿真时将覆盖它。

写入数据的数量和在哪些时间步写入数据由模块的参数确定。

（1）Decimation 参数，允许每 n 个采样写入一组数据，其中 n 是降采样因子。 缺省值为 1，表示每一步都写入数据。

（2）Sample time 参数，可以指定数据采集的采样间隔。这一参数在使用变步长求解器时是有用的，缺省值是-1，表示该模块在确定写入哪些点时从驱动模块那里继承采样时间。

数据类型接受双精度型实数信号。

参数　　Filename：指定放置矩阵的 mat 文件的名称。

　　　　Variable name：文件中包含的矩阵的名称。

　　　　Decimation：抽样因子，缺省值为 1。

　　　　Sample time：采集数据点的采样时间。

特点　　采样时间　　从驱动模块继承

　　　　向量化　　可以

6.2.5　To Workspace 模块

To Workspace 模块的属性对话框如图 6.23 所示。

Block Parameters: To Workspace
To Workspace
Write input to specified array or structure in MATLAB's main workspace.
Data is not available until the simulation is stopped or paused.
Parameters
Variable name:
simout
Limit data points to last:
inf
Decimation:
1
Sample time:
-1
Save format: Structure
OK　Cancel　Help　Apply

图6.23　To Workspace模块的属性对话框

说明：该模块将输入写入 MATLAB 工作空间中，由参数变量名（Varible name）指定的矩阵或结构中。参数保存格式（Save format）确定输出格式。

参数保存格式具体包括：

（1）矩阵

写入的数据数量以及在哪些时间步写入数据由模块的参数确定：

Limit data points to last 参数表明保存多少行数据，如果仿真产生的行数比指定的最大行数要多，仿真只保存最近产生的行，要保存所有数据，将该参数设置为 inf。

Sample time 参数，可以指定数据采集点的采样间隔。

Decimation 参数允许每 n 个采样写入一组数据，其中 n 是降采样因子，缺省值为 1，表示在每一步仿真时间步都写入数据。

在仿真期间，模块写数据到内部缓冲区，当仿真结束或者停止时，数据被写到工作空间。该模块的图标显示被写入数据的矩阵的名字。

（2）结构

这种格式的结构由三个字段组成：时间、信号和模块名。时间字段为空；模块名字段包含 To Workspace 模块的名字；信号字段包含一个两字段的结构：值和标示。其中值字段包含信号值矩阵。

（3）具有时间的结构

这一格式与结构格式相同，只是其时间字段包含仿真时间步向量。

（4）使用 From File 模块保存的数据。

如果用 To Workspace 模块写的数据，随后将被另一个仿真时用 From Workspace 模块重新输入，必须加入仿真时间的值。加进时间值的方法依赖于保存格式。

如果保存格式是结构，可以选择具有时间的结构来包含仿真时间作为保存格式，那么在输出结构中就保存有仿真时间向量。

如果保存格式是矩阵，必须加入一列仿真时间，方法有两种：

- 将 Clock 模块的输出作为多路输入给该模块的向量输入线的第一个元素；
- 通过仿真参数对话框的返回值或从命令窗口中指定时间。当仿真结束时，可以使用命令 matrix = [t; matrix]将时间向量 t 与矩阵合并成新的矩阵。

数据类型　可以将任何实数类型或复数类型的数据保存到 MATLAB 工作空间中。

参数　Variable name：保存数据的数组名。

　　Limit data points to last：保存的数据的最大数目，缺省值为 1000。

　　Decimation：抽样因子，缺省值为 1。

　　Sample time：采集数据点的采样时间。

　　Save format：保存数据的格式，缺省值为结构型（Structure）。

特点　采样时间　从驱动模块继承

　　向量化　可以

6.2.6 XY Graph 模块

XY Graph 模块的属性对话框如图 6.24 所示。

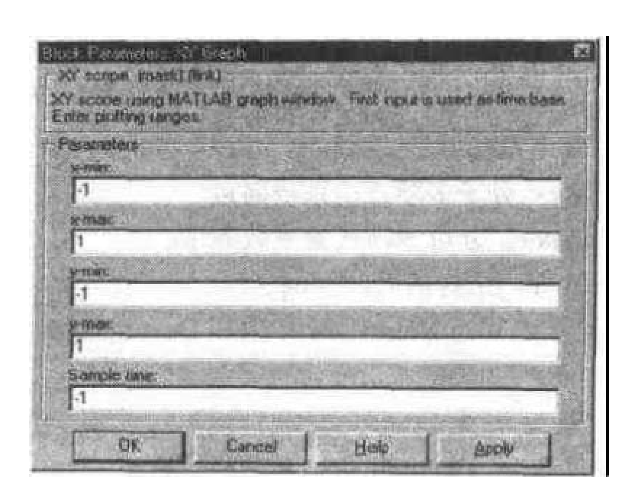


图 6.24　XY Graph 模块的属性对话框

说明：该模块在 MATLAB 的图形窗口中显示它的输入信号的 X-Y 曲线图。该模块有两个标量输入，模块绘制第一个输入的数据（X 轴方向）对第二个输入的数据（Y 轴方向）的曲线图。该模块对于检测两状态的数据很有帮助。超过指定范围的数据将不显示。

在仿真开始时，SIMULINK 会自动为每个 XY Graph 模块打开一个图形窗口。

要运行说明 XY Graph 模块的用法的演示，可以在命令窗口中输入 lorenzs，观看相应的例子。

数据类型　双精度型实数信号。

参数　x-min：x 轴最小值，缺省值为-1。

x-max：x 轴最大值，缺省值为 1。

y-min：y 轴最小值，缺省值为-1。

y-max：y 轴最大值，缺省值为 1。

Sample time：采样间隔时间，缺省值为-1，即采样时间由其驱动模块决定。

特点　采样时间　从驱动模块继承

状态个数　0

6.3　Discrete 库

6.3.1　Discrete Filter 模块

Discrete Filter 模块的属性对话框如图 6.25 所示。

说明：该模块实现无限脉冲响应（IIR）和有限脉冲响应（FIR）滤波器，可以使用 Numerator 和 Denominator 参数以向量的形式指定分子和分母的 z^{-1} 的升幂多项式的系数。分母的阶数必须大于或者等于分子的阶数。

Discrete Filter 模块采用的是信号处理人员经常使用的方法，即使用 z^{-1} 多项式描述数字滤波器。而 Discrete Transfer Fcn 模块代表控制工程人员经常使用的方法，他们使用 z 的多项式描述离散系统，当分子和分母的长度相等时，两种方法是等效的。一个有 n 个元素的向量描

述一个 n-1 阶多项式。

该模块能够根据指定的分子分母，在模块的图标中显示相应的分子分母表达式。

数据类型　接受和输出双精度型实数信号。

参数　Numerator：分子多项式的系数向量。

Denominator：分母多项式的系数向量。

Sample time：采样的时间间隔。

特点

直通特性	当 Numerator 参数和 Denominator 参数的参数相等时存在
采样时间	离散
标量扩展	无
状态个数	Denominator 参数的长度
向量化	无
零点穿越	无

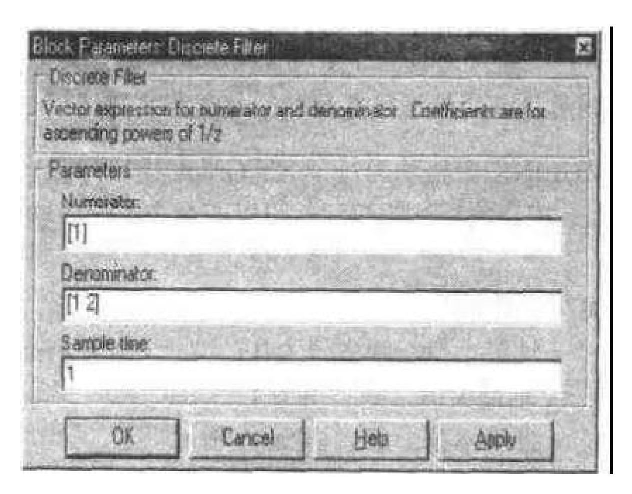


图 6.25　Discrete Filter 模块的属性对话框

6.3.2　Discrete State-Space 模块

Discrete State-Space 模块的属性对话框如图 6.26 所示。

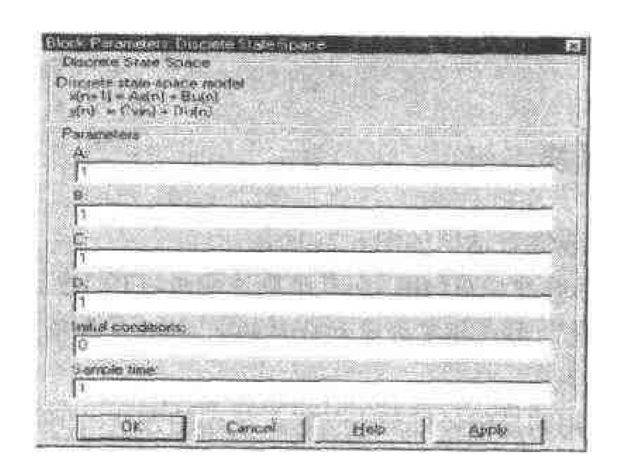


图 6.26　Discrete State-Space 模块的属性对话框

该模块实现由式（6.1）描述的系统

$$
\begin{aligned}
x(n+1) &= Ax(n) + Bu(n) \\
y(n) &= Cx(n) + Du(n)
\end{aligned} \tag{6.3}
$$

其中，u 是输入，x 是状态，y 是输出。该模块接受一个输入并且产生一个输出，输入向量的宽度由矩阵 B 和 D 的列数确定。输出向量的宽度由矩阵 C 和 D 的行数确定。

SIMULINK 可以将包含有较多 0 的矩阵转换为稀疏矩阵来实现快速乘法。

数据类型　接受双精度型实数信号。

参数　A, B, C, D：状态方程的系数矩阵。

Initial conditions：初始状态向量，缺省值为 0。

Sample time：采样时间间隔。

特点

直通特性	当 D≠0 时存在直通性质
采样时间	离散
标量扩展	初始条件存在标量扩展
状态个数	由 A 矩阵的大小决定
向量化	可以
零点穿越	无

6.3.3　Discrete-Time Integrator 模块

Discrete-Time Integrator 模块的属性对话框如图 6.27 所示。

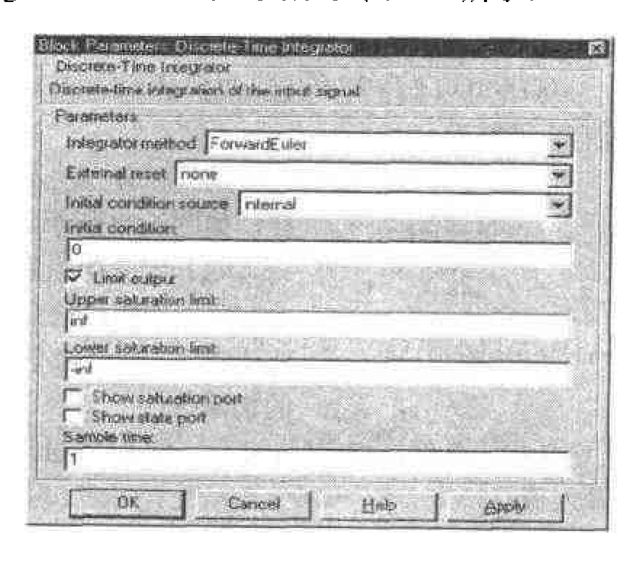


图 6.27　Discrete-Time Integrator 模块的属性对话框

说明：在构造一个纯离散系统时，可以用该模块替代 Integrator 模块。

通过该模块可以在模块的对话框中或者模块的输入中定义初始状态；输出模块状态；定义积分的上下限；根据另外的一个复位输入对状态复位。

（1）积分方法

该模块使用的积分方法有：前向欧拉法、后向欧拉法和梯形法。对于给定的积分步 k，SIMULINK 更新 y（k）和 x（k+1）。T（触发情况下采样时间 ΔT）是采样周期。值可以根据

上下限进行删减。

- 前向欧拉方法(缺省方法)，又称前向矩形法或者左手近似。对于这种方法，用 $T/(z-1)$ 近似 $1/s$，这样可以得到：$y(k)=y(k-1)+T*u(k-1)$。

假定 $x(k)=y(k)$ 得到：

$$x(k+1)=x(k)+T*u(k)$$
$$y(k)=x(k)$$

使用此方法，输入端口 1 没有直通特性。

- 后向欧拉法，也叫后向矩形法或者右手近似。对于这种方法，用 $T*z/(z-1)$ 来近似 $1/s$。这样得到：$y(k)=y(k-1)-T*u(k)$。

假定 $x(k)=y(k-1)$，得到：

$$x(k+1)=y(k)$$
$$y(k)=x(k)+T*u(k)$$

使用此方法，输入端口 1 具有直通特性。

- 梯形法，对于这种方法，用 $T/2*(z+1)/(z-1)$ 近似 $1/s$。这样得到

$$y(k)=y(k-1)+T/2*(u(k)+u(k-1))$$

当 T 固定时（等于采样周期），让 $x(k)=y(k-1)+T/2*u(k-1)$，得到：

$$x(k+1)=y(k)+T/2*u)k)$$
$$y(k)=x(k)+T/2*u(k)$$

其中，$x(k+1)$ 是下一个输出的最佳估计。在 $x(k)!=y(k)$ 的意义上，它不正好是状态。

当 T 可变时（也就是说，它由触发时间得到），得到：

$$x(k+1)=y(k)$$
$$y(k)=x(k)+T/2*(u(k)+u(k-1))$$

其中，输入端口 1 具有直通特性。

选择积分方法不同，模块的图标也不一样，如图 6.28 所示。

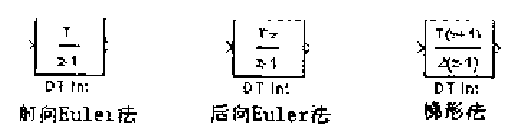


图6.28　Discrete-Time Integrator模块的属性对话框

（2）定义初始条件

可以在模块的对话框中以一个参数的形式或者以一个外部信号输入的形式定义模块的初始条件。

要以模块参数的形式定义初始条件，指定Initial condition source参数为internal，并且在Initial condition参数域中输入值。

要从一个外部信号源提供初始状态，指定Initial condition source参数为external。这时在模块输入端口的下部出现另外一个输入端口。

（3）使用状态端口

在两种已知情况下，必须使用状态端口而不是输出端口。

模块的输出通过复位端口或者初始状态端口反馈给模块，形成了一个反馈回路。如果想将状态从一个条件执行的子系统传给另外一个时，这时会出现时间上的问题，要解决这个问题，可以将状态通过状态端口而不是输出端口传送。尽管值相同，SIMULINK产生它们的时间有细微的差别，这就保证了模型不会出现这些问题。通过选择Show state port选择框，可以输入模块的状态。

缺省情况下，状态端口显示在模块的上端。

（4）限制积分

为了防止输出超出可指定的范围，选择Limit output选择框并在合适的参数域中输入限制参数，这样做使得该模块在功能上像一个有限积分器。当输出超出了限制时，积分过程被关断以防止积分结束。在仿真期间，可以改变这些限制参数，但不能改变输出是否是有限的。输出由下列因素确定：

当积分结果小于Lower saturation limit，并且输入为负时，输出保持在Lower saturation limit。

当积分结果介于Lower saturation limit与Upper saturation limit之间时，输出积分结果。

当积分结果大于Upper saturation limit，并且输入是正数时，输出保持在上饱和限。

要生成一个信号以能够表明其状态什么时候是受限的，选择Show saturation port选择框，这时一个饱和端口出现在模块输出端口的底端。

1：表明积分超出了饱和上限；

0：表明积分没有超限；

-1：表明积分积分超出了饱和下限。

当选择了Limit output选项，模块有三个过零区间：一个检测它何时达到饱和上限；一个检测何时达到了饱和下限；另外一个检测何时离开饱和状态。

（5）复位状态

模块可以通过一个外部信号复位状态为指定的初始条件。要使得模块重置它的状态，选择External reset选项。一个触发端口显示在模块输入端口的下方并且显示了其触发类型。

当触发信号有上升沿时，选择rising触发状态复位。

当触发信号有下降沿时，选择falling触发状态复位。

当触发信号既有上升沿又有下降沿时，选择either触发状态复位。

复位端口是直通的，如果模块的输出，直接或通过一系列模块直通给了这一端口，将会导致代数回路。要解除这一回路，将状态端口传给复位端口。要获得模块的状态，选择Show state port选项框。

（6）选择所有选项。选择所有选项后，图标如图6.29所示。

图 6.29　离散时间积分器模块在图标中显示端口

数据类型　接受和输出双精度类型实数信号。

参数　Integrator method：积分方法。缺省值为 ForwardEuler。

External reset：当复位信号触发时间发生时，复位其状态为初始条件。

Initial condition source：指定获得初始条件来自 Initial condition 参数（设为 interal），还是外部模块（设为 external）。

Initial condition：在初始条件源选 internal（内部）是，指状态初始条件。

Limit output：如果选择该选择框，将限制输出状态值在饱和下限和饱和上限参数之间。

Upper saturation limit：积分上限，缺省值为 inf。

Lower saturation limit：积分下限，缺省值为-inf。

Show saturation port：如果选择该选择框，将在模块上加一个饱和输出端口。

Show state port：如果选择该选择框，将在模块上加一个状态输出端口。

Sample time：采样时间间隔，缺省值为 1。

特点	直通	当复位和外部初始条件源端口时
	采样时间	离散
	标量扩展	参数扩展
	状态个数	由驱动模块和参数模块继承
	向量化	可以
	零点穿越	检测复位有一个过零区间；检测上、下饱和限各有一个过零区间，离开饱和有一个过零区间。

6.3.4　Discrete Transfer Fcn 模块

Discrete Transfer Fcn 模块的属性对话框如图 6.30 所示。

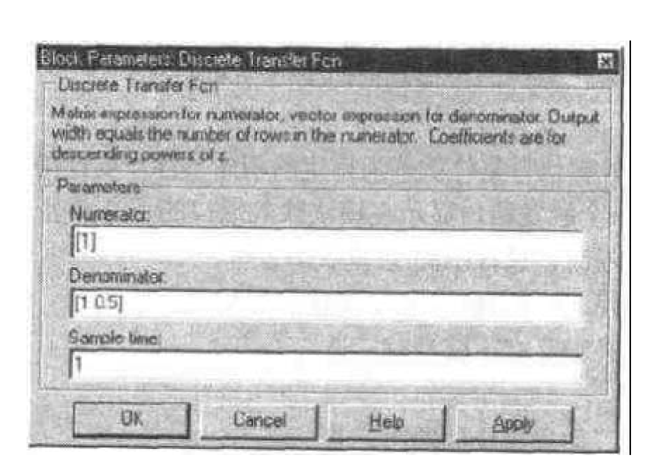


图 6.30　Discrete Transfer Fcn 模块的属性对话框

实现（6.3）式描述的 z 变化传递函数：

$$H(z)=\frac{mun(z)}{den(z)}=\frac{num_0 z^n+num_1 z^{n-1}+\cdots+unm_m z^{n-m}}{den_0 z^n+den_1 z^{n-1}+\cdots+den_n} \tag{6.4}$$

其中 m+1 和 n+1 分别是分子和分母系数的个数，num 和 den 按 z 的降幂包含分子和分母的系数。num 可以是向量也可以是矩阵，但 den 必须是向量，可以在模块的对话框中指定这些参数。分母的阶数必须大于等于分子的阶数。

模块的输入是标量，输出宽度等于分子的行数。

该模块代表了控制工程人员常用的典型方法，他们用 z 的多项式描述离散系统。而 Discrete Filter 模块代表了信号处理人员用的典型方法，他们用 z^{-1} 的多项式描述数字滤波器，当分子分母的阶数相等时，这两种方法相同。

该模块在图标中显示了它是如何被指定的。

数据类型　　接受双精度实数信号。

参数	Numerator：分子多项式的行向量，缺省值为[1]。	
	Denominator：分母多项式的行向量，缺省值为[1 0.5]。	
	Sample time：模块的采样时间，缺省值为 1。	
特点	直通	当 Numerator 与 Denominator 参数长度相等时
	采样时间	离散
	标量扩展	无
	状态个数	由 Denominator 参数的长度决定
	向量化	无
	零点穿越	无

6.3.5 Discrete Zero-Pole 模块

Discrete Zero-Pole 模块的属性对话框如图 6.31 所示。

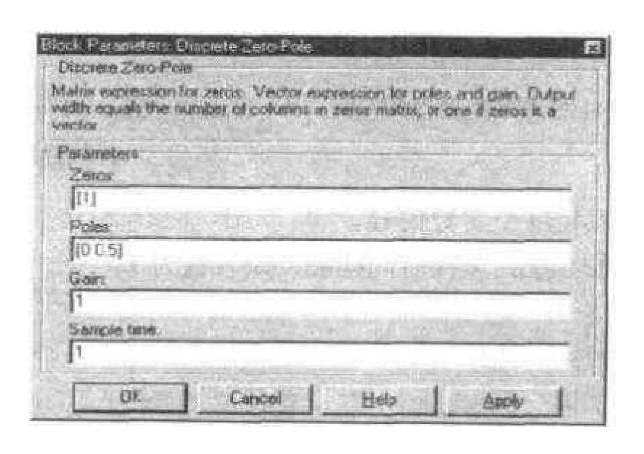


图 6.31 Discrete Zero-Pole 模块的属性对话框

说明：该模块实现一个用延迟因子 z 的零点、极点和增益形式给出的离散系统。

一个传递函数可以用零点-极点-增益的形式进行表达，对于 MATLAB 中的一个单输入单输出的系统，它们之间的关系可以用（6.5）式进行描述：

$$H(z)=K\frac{Z(z)}{P(z)}=K\frac{(z-Z_1)(z-Z_2)\cdots(z-Z_m)}{(z-P_1)(z-P_2)\cdots(z-P_n)} \tag{6.5}$$

其中，Z 表示零点向量，P 表示极点向量，K 表示增益。极点的数目必须大于或等于零点的数目。如果零点和极点是复数，则必须是共轭的。

参数	Zeros：零点矩阵，缺省值为 1。	
	Poles：极点矩阵，缺省值为[0 0.5]。	
	Gain：增益矩阵，缺省值为 1。	
	Sample time：采样时间间隔，缺省值为 1。	
特点	直通	当零极点个数相等时
	采样时间	离散
	标量扩展	无
	状态个数	由 Poles 向量的长度决定

向量化　　无
零点穿越　无

6.3.6　First-Order Hold 模块

First-Order Hold 模块的属性对话框如图 6.32 所示。

说明：该模块实现以一定的指定采样间隔执行的一阶采样保持。

可以通过运演示程序 fohdemo 来观察零阶保持和一阶保持之间的区别。图 6.33 比较了 Sine Wave 模块和从 First-Order Hold 模块模块的输出结果。

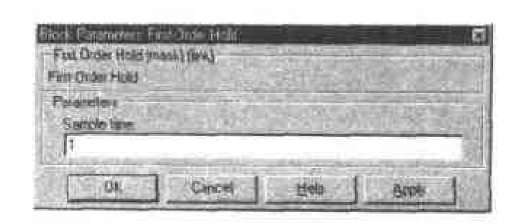


图 6.32　First-Order Hold 模块的属性对话框

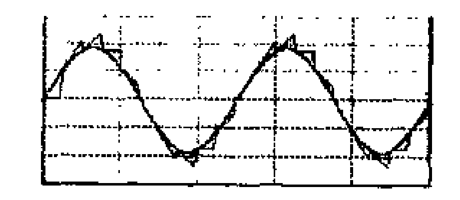


图 6.33　Sine Wave 模块和 First-Order Hold 模块输出结果比较

数据类型　支持双精度型实数信号。
参数　Sample time：采样时间间隔，缺省值为 1。
特点　直通　无
采样时间　连续
标量扩展　无
状态个数　对于每个输入都包括一个离散状态和一个连续状态
向量化　可以
零点穿越　无

6.3.7　Zero-Order Hold 模块

Zero-Order Hold 模块的属性对话框如图 6.34 所示。

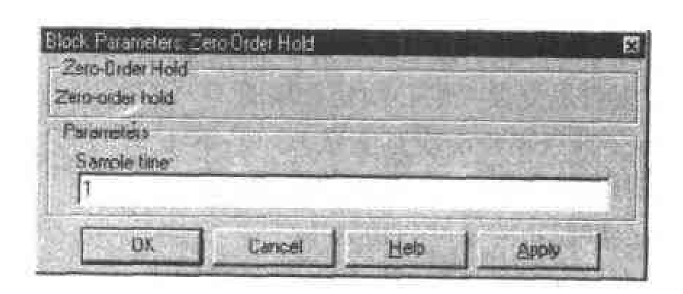


图 6.34　Zero-Order Hold 模块的属性对话框

说明：该模块实现指定采样速率的采样和保持功能。它有一个输入和输出端口，输入和输出信号可以是标量也可以是向量。

该模块可用于一个或多个信号进行离散化或以另外的速率对信号进行重新采样。如果需要模拟采样，但不需要另外的更为复杂的离散功能模块时，可以使用这一模块。例如，用它连接 Quantizer 模块以及模拟对输入有放大作用的 A/D 转换器。

数据类型	支持任何类型的实数或复数信号。	
参数	Sample time：模块的采样时间。	
特点	直通	有
	采样时间	离散
	标量扩展	有
	状态个数	0
	向量化	可以
	零点穿越	无

6.3.8 Unit Delay 模块

Unit Delay 模块的属性对话框如图 6.35 所示。

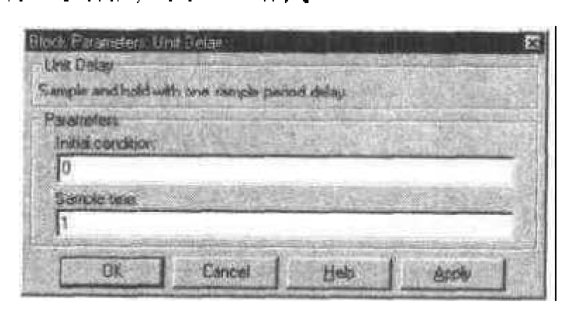


图 6.35 Unit Delay 模块的属性对话框

说明：该模块将它的输入信号延迟并保持一个采样间隔。如果模块的输入是向量，向量中所有元素的延迟时间都相同。该模块与离散时间算子 z-1 的作用相同。

如果需要无延迟的采样保持函数，可以使用零阶保持器（Zero-Order Hold）模块，如果需要大于一个单位的延迟，可以使用离散传递函数（Discrete Transfer Fcn）模块。

数据类型接受和输出任何类型的实数或复数信号，包括用户自定义数据类型，此时初始条件必须为零。

参数 Initial condition：第一个仿真周期的模块输出。该参数需要仔细选择，缺省值为 0。
Sample time：采样时间，缺省值为 1 秒。

6.4 Continuous 库

6.4.1 Derivative 模块

Derivative 模块的属性对话框如图 6.36 所示。

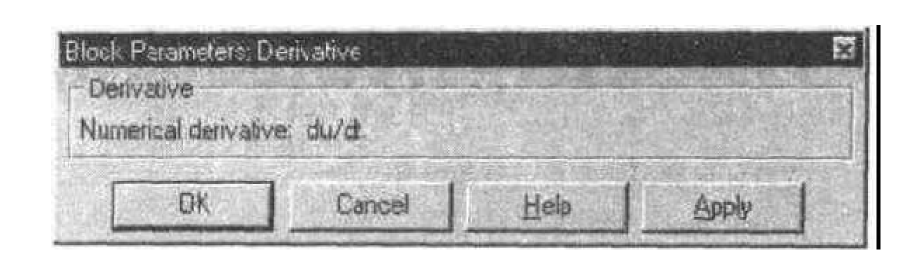


图 6.36　Derivative 模块的属性对话框

该模块近似地给出其输入的导数，计算公式

$$\frac{\Delta u}{\Delta t} \tag{6.6}$$

其中，Δu 是输入值得变化，Δt 是从上一仿真时间步到该步的时间变化。该模块接受一个输入并且产生一个输出。在仿真开始之前输入信号的值被认为是 0。模块的初始输出是 0。

输出结果的精度取决于各个仿真步所花的时间。时间步越短，该模块的输出曲线将会越来越光滑，结果越精确。与有连续状态的模块不同，该模块的求解器在输入变化比较快时不会使用较短的时间步。

但输入是离散信号且输入改变时，其连续导数为脉冲，否则为 0。可以使用式（6.7）得到一个离散信号的离散导数：

$$y(k)=\frac{1}{\Delta t}(u(k)-u(k-1)) \tag{6.7}$$

将（6.6）式作 Z 变化，得到：

$$\frac{Y(z)}{U(z)}=\frac{1-z^{-1}}{\Delta t}=\frac{z-1}{z\cdot\Delta t} \tag{6.8}$$

使用 linmod 函数给一个包含有 Derivative 模块的模型进行线性化，将会遇到困难。

数据类型双精度类型实数信号：

参数	直通	有
	采样时间	连续
	标量扩展	不适用
	状态个数	0
	向量化	可以
	零点穿越	无

6.4.2　Integrator 模块

Integrator 模块的属性对话框如图 6.37 所示。

说明：该模块对其输入信号进行积分。积分器的输出仅仅是其状态（积分）。通过该模块可以在模块的对话框中或作为模块的输入定义为其初始状态；输出模块状态；定义积分结果的上限和下限；通过另外的一个复位输入对状态复位。

当构造一个纯离散系统时，应使用离散系统积分（Discrete –Time Integrator）模块。

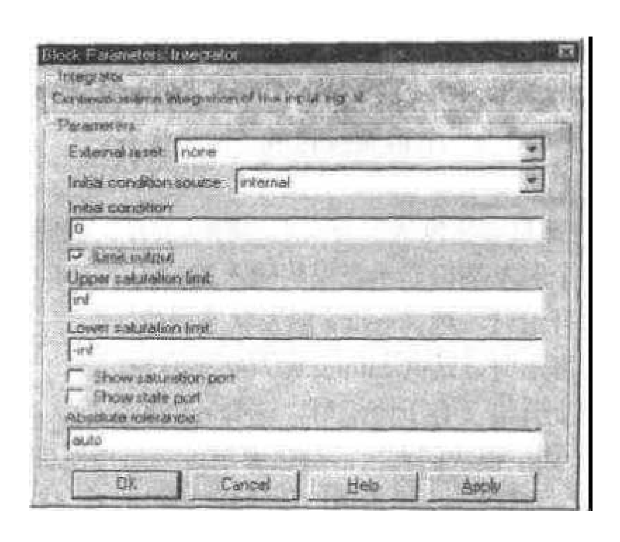


图 6.37 Integrator 模块的属性对话框

（1）定义初始状态

可以通过模块的属性对话框中的参数或从外部输入一个信号来定义模块的初始状态。

以模块的参数的形式定义初始状态：指定 internal 作为 Initial condition source 参数的值，并在 Initial condition 参数域中输入数值。

以一个外部信号源的形式提供初始状态：指定 external 作为 initial condition source 参数的值。在模块输入的下部会显示另外一个输入端口：初始条件输入端口。

（2）使用状态端口

在两种情况已知的情况下，必须使用状态端口而不是输出端口。

当模块的输出通过复位端口或者初始状态端口反馈给模块，这会将导致出现代数环。

如果想将状态从一个条件执行的子系统传给另一个条件执行子系统时，则会导致时间问题。

如果通过将状态从状态端口而不是输出端口传出来纠正这些问题。尽管数值是相同的，SIMULINK 产生它们的时间略有不同，这样保证了系统不会出现这些问题。可以通过选取 Show state port 选择框输出模块的状态。缺省情况下，状态端口显示在模块的顶部。

（3）限制积分结果

为了显示输出模块的状态，选择Limit output选择框，并在适当的参数域中输入限制参数。这样做使得该模块在功能上像一个有限积分器。当输出超出了限制时，积分过程被关断以防止积分结束。在仿真期间，可以改变这些限制参数，但不能改变输出是否是有限的。输出由下列因素确定：

当积分结果小于Lower saturation limit，并且输入为负时，输出保持在Lower saturation limit。

当积分结果介于Lower saturation limit与Upper saturation limit之间时，输出积分结果。

当积分结果大于Upper saturation limit，并且输入是正数时，输出保持在上饱和限。

要生成一个信号以能够表明其状态什么时候是受限的，选择Show saturation port选择框，这时一个饱和端口出现在模块输出端口的底端。

1：表明积分超出了饱和上限；

0：表明积分没有超限；

-1：表明积分超出了饱和下限。

当选择了Limit output选项，模块有三个过零区间：一个检测它何时达到饱和上限；一个检测何时达到了饱和下限；另外一个检测何时离开饱和状态。

（4）复位状态

模块可以通过一个外部信号复位状态为指定的初始条件。要使得模块重置它的状态，选择External reset选项。一个触发端口显示在模块输入端口的下方并且显示了其触发类型。

- 当触发信号有上升沿时，选择 rising 触发状态复位。
- 当触发信号有下降沿时，选择 falling 触发状态复位。
- 当触发信号既有上升沿又有下降沿时，选择 either 触发状态复位。

复位端口是直通的，如果模块的输出，直接或通过一系列模块直通给了这一端口，将会导致代数回路。要解除这一回路，将状态端口传给复位端口。要获得模块的状态，选择Show state port选项框。

（5）指定模块状态的绝对容许误差

如果模型当中包含有幅度变化很大的状态时，为模型定义绝对容许误差不可能提供有效的误差控制。要定义一个 Integrator 模块的状态的绝对值容限。在 Absolute tolerance 参数域中输入一个数值。如果模块有多个状态，同以数值适用于所有状态。

（6）选择所有项

数据类型　数据端口接受和输出双精度类型的实数信号，而外部复位端口接受双精度或逻辑类型的信号。

参数

External reset：当触发事件发生时，复位状态为初始状态。

External reset：当复位信号触发时间发生时，复位其状态为初始条件。

Initial condition source：指定获得初始条件来自 Initial condition 参数（设为 interal），还是外部模块（设为 external）。

Initial condition：在初始条件源选 internal（内部）是，指状态初始条件。

Limit output：如果选择该选择框，将限制输出状态值在饱和下限和饱和上限参数之间。

Upper saturation limit：积分上限，缺省值为 inf。

Lower saturation limit：积分下限，缺省值为-inf。

Show saturation port：若选择该选择框，将在模块上加一个饱和输出端口。

Show state port：如果选择该选择框，将在模块上加一个状态输出端口。

Absolute tolerance：模块的绝对容限。

特点

直通	外部初始条件源复位端口，有直通特性。
采样时间	连续
标量扩展	参数扩展
状态个数	从驱动模块继承
向量化	可以
零点穿越	如果限制输出，有过零区间，检测复位有一个；上下限饱和限各一个，离开饱和一个。

6.4.3 Memory 模块

Memory 模块的属性对话框如图 6.38 所示。

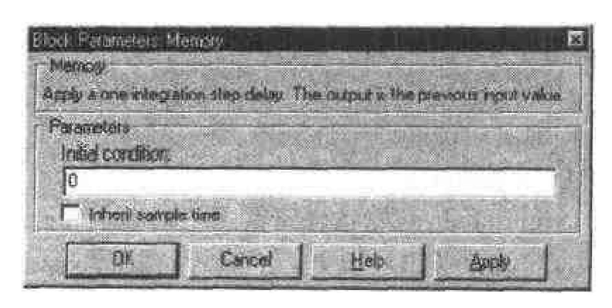


图 6.38 Memory 模块的属性对话框

说明：该模块输出它的前一积分步的输入，对它的输入信号使用一个积分步的采样和保持。

当使用 ode15s 或者 ode113 积分时，应避免使用 Memory 模块，除非模块的输入保持不变。

数据类型　支持任何类型的实数或复数信号，包括用户的自定义类型，如果输入为用户的自定义类型，则初始条件必须为零。

参数　Initial condition：指初始积分步的输出。

Inherit sample time：选中该选择框，采样时间将从驱动模块继承。

特点
- 直通　无
- 采样时间　连续，如果选中 Inherit sample time 参数，采样时间将从驱动模块继承。
- 标量扩展　初始条件参数扩展
- 状态个数　0
- 向量化　可以
- 零点穿越　无

6.4.4 State-Space 模块

State-Space 模块的属性对话框如图 6.39 所示。

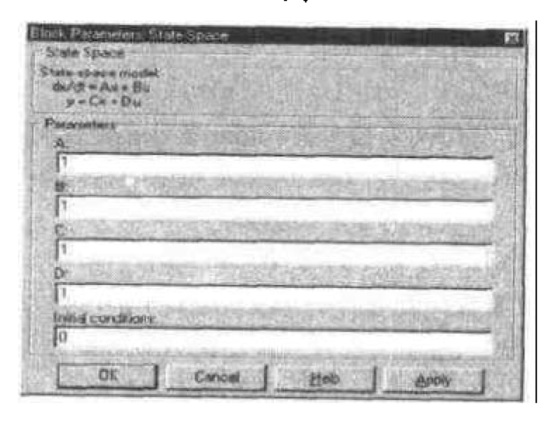


图 6.39 State-Space 模块的属性对话框

说明：该模块实现式（6.9）定义的系统

$$x'=Ax+Bu$$
$$y=Cx+Du \tag{6.9}$$

其中，x 是状态向量，u 是输入向量，y 是输出向量。该模块接受一个输入并且产生一个输出，输入向量的宽度由 B 和 D 矩阵的行数确定。

SIMULINK 将含有 0 的矩阵转换为稀疏矩阵，以实现更有效的乘法。

数据类型　接受和输出双精度型实数信号。

参数　A,B,C,D 为系数矩阵

Initial conditions：初始状态向量。

特点

直通	当矩阵 D≠0 时
采样时间	连续
标量扩展	初始条件参数扩展
状态个数	由矩阵 A 的大小决定
向量化	可以
零点穿越	无

6.4.5　Transfer Fcn 模块

Transfer Fcn 模块的属性对话框如图 6.40 所示。

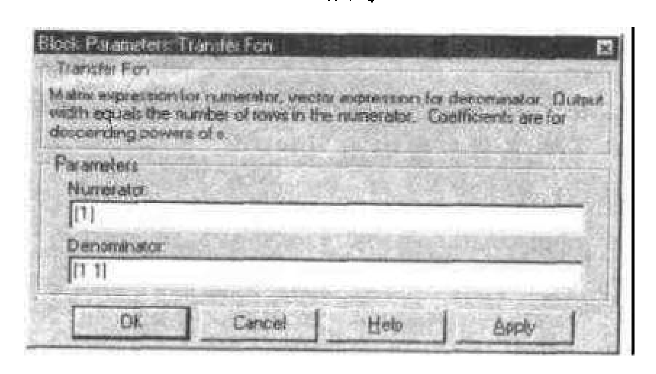


图 6.40　Transfer Fcn 模块的属性对话框

该模块实现一个传递函数，它的输入 u 与输出 y 可以用传递函数表示

$$H(s)=\frac{y(s)}{u(s)}=\frac{num(s)}{den(s)}=\frac{num(1)s^{nn-1}+num(2)s^{nn-2}+\cdots+num(nn)}{den(1)s^{nn-1}+den(2)s^{nn-2}+\cdots+den(nd)} \tag{6.10}$$

其中，nn,nd 分别为分子分母的系数个数。num 和 den 是以 s 的降幂表示的分子和分母的系数。num 可以是向量也可以是矩阵，den 必须是向量，它们都可以在对话框的相应参数中指定。分母的阶数必须大于或者等于分子的阶数。

模块输入是标量，输出的宽度等于分子中的行数。

初始状态预设为 0。如果需要指定初始状态，使用 uf2ss 将传递函数转换成状态空间的形式并且使用 State-Space 模块。tf2ss 函数为系统提供 A,B,C,D 矩阵。

显示在该模块图标中的分子和分母的形式取决于它们是如何指定的。

如果都指定为表达式、向量或者用圆括号包括的变量，图标显示用指定的系数和 s 的幂组成的传递函数。如果指定为用圆括号包括的变量，变量的值会被计算出来。例如，如果指

定 Numerator 为[3,2,1]，Denominator 为(den)，其中，den 是[7,5,3,1]，模块的图标如图 6.41(a)所示。



图 6.41　分子分母形式不同的两种图标

如果都是指定为变量，图标显示后跟“(s)”的变量名，例如，如果指定 Numerator 为 num，Denominator 为 den，模块的图标如图 6.41(b)所示。

数据类型　　接受和输出任何数据类型的信号。

参数　　Numerator：分子系数多项式，一个具有多行的矩阵可以用来指定产生多个输出，缺省值为[1]。

　　Denominator：分母系数行向量。缺省值为[1 1]。

特点　　直通　　当 Numerator 和 Denominator 参数长度相等时

　　采样时间　连续

　　标量扩展　无

　　状态　　由 Denominator 的参数的长度减 1

6.4.6　Transfer Delay 模块

Transfer Delay 模块的属性对话框如图 6.42 所示。

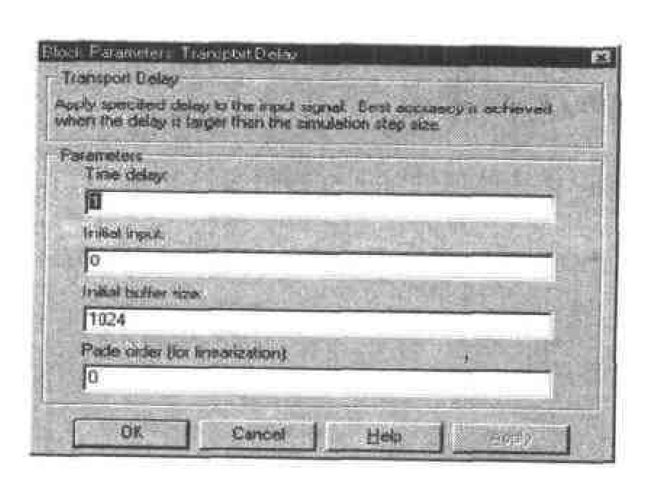


图 6.42　Transfer Delay 模块的属性对话框

说明：该模块将输入延迟一段指定的时间，可以用来模拟时间延迟。

在仿真开始时，模块输出 Initial input 参数值直到仿真时间超过 Time delay 参数值，这时模块开始产生经过延迟的输入，Time delay 参数必须是非负数。

该模块在仿真期间在缓冲区中保存输入的数据点的数目超过了缓冲区的大小，模块分配另外的内存并且在仿真结束后 SIMULINK 显示一条消息以表示一共需要多大的缓冲区。因为分配另外的内存会降低仿真的速度，如果仿真速度很重要，则必须仔细的设定该参数值。对

于长时间的延迟，该模块会需要大量的内存，特别是对于向量化的输入。

当需要一个时刻的输出，而这一时刻与保存的输入值的时间点没有对应值，这时模块在各个点间进行线性插值。当延迟小于步长时，模块从最后的输入点外推，这可能会得到不精确的结果，因为该模块没有直通特性，因此不能使用当前的输入计算它的输出值。要说明这一点，考虑一个步长为 1 的定步长且当前时间 t=5 的仿真。如果延迟是 0.5，该模块需要生成 t=4.5 的输入点，因为最近保存的时间值是 t=4，因此模块进行前向外推。

该模块对离散信号不进行插值。它返回 t-tdelay 时的离散值。

该模块与 Unit Delay 模块不同，Unit Delay 模块只在采样点处延迟并保持输出。

使用 linmod 线性化包含有 Transfer Delay 模块的模型可能会遇到错误。

数据类型　　接受和输出双精度的实数信号

参数　　Time delay：输入信号在传给输出前被延迟的仿真时间量，该参数值不能为负数，缺省值为 1。

Initial input：指模块在仿真开始与时间延迟之间产生的输出。缺省值为 0。

Initial buffer size：初始分配内存存贮的点数。缺省值为 1024。

Padeorder (for linearization)：线性近似的阶数，如果为零，表示在线性化时只保留标量，没有动态状态。如果取正数 n，则模块相应增加 n 个状态，不过线性化后的模型精度也会相应提高。

特点

直通	无
采样时间	连续
标量扩展	除初始缓存大小之外的所有参数和输入有标量扩展
向量化	可以
零点穿越	无

6.4.7　Variable Transport Delay 模块

Variable Transport Delay 模块的属性对话框如图 6.43 所示。

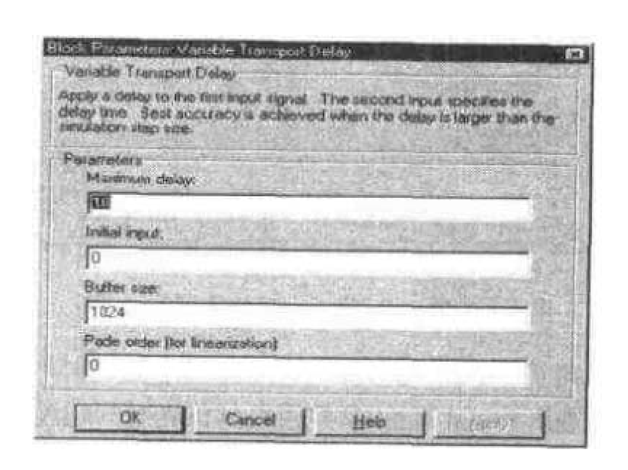


图 6.43　Variable Transport Delay 模块的属性对话框

说明：该模块可以用来模拟可变时间的延迟。例如可以用来模拟管道的系统，管道中泵中液体的速度是可变的。

该模块有两个输入：第一个是传过模块的信号，第二个是时间延迟。

参数最大延迟（Maximum delay）定义对输入的最大延迟时间。该模块截去超过该值的延迟时间值，最大延迟必须大于或者等于零。如果延迟时间变成负数，模块将它截为零，并给出警告信息。

在仿真期间，该模块保存成对的时间和输入值到内部缓冲区。在仿真开始的时候，模块的输出是 Initial input 设置的值，直到仿真时间超过输入的延迟时间。然后在每一仿真步，模块输入的信号是当前仿真时间减去延迟时间时的输入信号。

当需要某一个时刻的输出，而保存的时间/输出值中没有对应的时间，模块在数据点之间线性插值。如果时间延迟小于步长，模块外推得到输出点，这时结果可能不是很准确。该模块不能使用当前的输入值去计算它的输出值，因为它不是直通的。

该模块对离散信号不进行插值。它返回 t-tdelay 时的离散值。

该模块与 Unit Delay 模块不同，Unit Delay 模块只在采样点处延迟并保持输出。

数据类型　接受和输出双精度类型实数信号。

参数　Maximum delay：时间延迟输入的最大值。该值不能为负。缺省值为 10。

Initial input：仿真的第一次超过时间延迟输入之前，模块产生的输出。缺省值为 0。

Buffer size：模块可存储点数。缺省值为 1024。

Padeorder (for linearization)：线性近似的阶数，如果为零，表示在线性化时只保留标量，没有动态状态。如果取正数 n，则模块相应增加 n 个状态，不过线性化后的模型精度也会相应提高。

特点　直通　时间延迟（第二个输入）是直通的

采样时间　连续

标量扩展　除初始缓存大小之外的所有参数和输入有标量扩展

向量化　可以

零点穿越　无

6.4.8 Zero-Pole 模块

Zero-Pole 模块的属性对话框如图 6.44 所示。

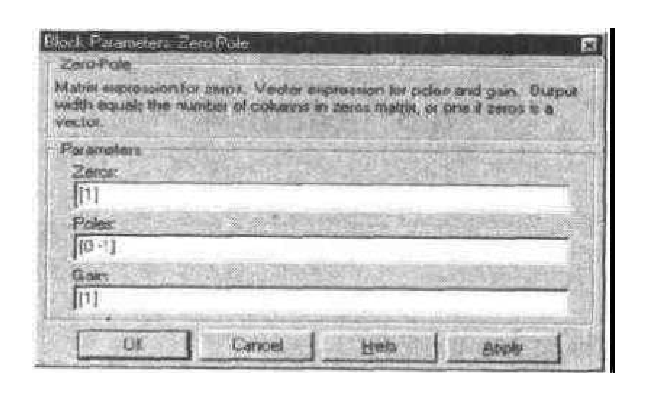


图 6.44　Zero-Pole 模块的属性对话框

说明：该模块可以实现以零极点和增益方式表示的系统。

该系统以（6.11）式表示

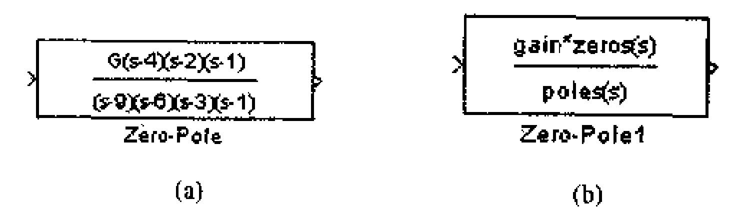
$$H(s)=K\frac{Z(s)}{P(s)}=K\frac{(s-Z(1))(s-Z(2))\cdots(s-Z(m))}{(s-P(1))(s-P(2))\cdots(s-P(n))} \tag{6.11}$$

其中，Z 表示零点向量，P 表示极点向量，K 为增益。Z 可以是向量也可以是矩阵，P 必须是向量，K 可以是标量或者是长度等于 Z 的行数的向量。极点的个数必须大于或者等于零点的个数。如果零极点是复数，它们必须是共轭负数对。

模块的输入输出宽度等于零点矩阵的行数。Zero-Pole 模块在其图标中根据指定的参数来显示它的传递函数。如果每一个参数都被指定为表达式或者向量，图标显示用指定的零点、极点和增益表示的传递函数，如果参数被指定为变量（用括号表示），图标中将显示变量的值。

例如，如果 Zeros 参数设置为[4,2,1]，Poles 参数被指定为（poles），而 poles 在工作空间中被定义为[9,6,3,1]，Gain 参数设置为 G，图标将如图 6.45(a)所示。

如果 Zeros 参数设置为 zeros，Poles 参数被指定为 poles，而 poles 在工作空间中被定义为[9,6,3,1]，Gain 参数设置为 gain，图标将如图 6.45(b)所示。



(a)　　(b)

图 6.45　Zero-Pole 模块的两种不同图标

参数	Zeros：系统零点矩阵，缺省值为[1]。	
	Poles：系统极点向量，缺省值为[0　−1]。	
	Gain：系统增益向量，缺省值为[1]。	
特点	直通	当系统零极点参数长度相等时有直通
	采样时间	连续
	标量扩展	无
	状态个数	由极点长度决定
	向量化	无
	零点穿越	无

6.5　Math 库

6.5.1　Abs 模块

Abs 模块的属性对话框如图 6.46 所示。

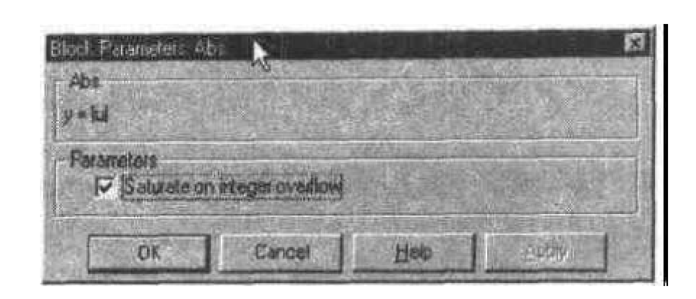


图 6.46　Abs 模块的属性对话框

说明：该模块产生的输出是输入的绝对值，它接受一个输入并产生一个输出。

数据类型　接受任何类型的实数或复数值输入，产生一个与输入同样类型的实数输出。

参数　　Saturate on integer overflow：缺省时为选中状态，这时该模块将最小的负数对应到相应的最大正数，如

8 位整型　　-128 映射到 127

16 位整型　　-32768 映射到 32767

32 位整型　　-2147483648 映射到 2147483647

如果该选项不被选中，则该模块在最小负数输入下的行为将无法预知。

特点　　直通　　有

采样时间　　由驱动模块继承

标量扩展　　不适用

向量化　　可以

零点穿越　　有

6.5.2　Algebraic Constraint 模块

Algebraic Constraint 模块的属性对话框如图 6.47 所示。

该模块将输入信号 f(z)约束为 0，并输出一个代数的状态 z。该模块输出使输入为零时所必须的值。输入必须通过一些反馈影响输入。这样就可以为一阶微分/代数系统（DAEs）指定代数方程。

缺省情况下，参数 Initial guess（初始估计）的值为 0。可以通过给代数状态 z 提供一个接近于计算结果的初始值来提高代数回路计算的效率。

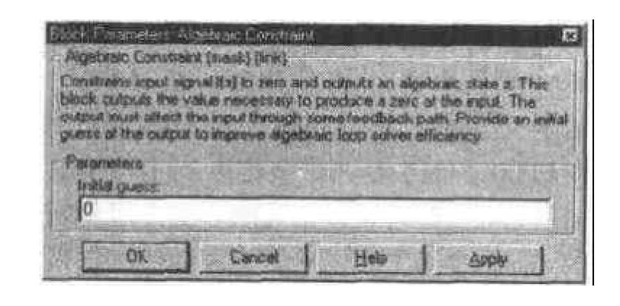


图 6.47　Algebraic Constraint 模块的属性对话框

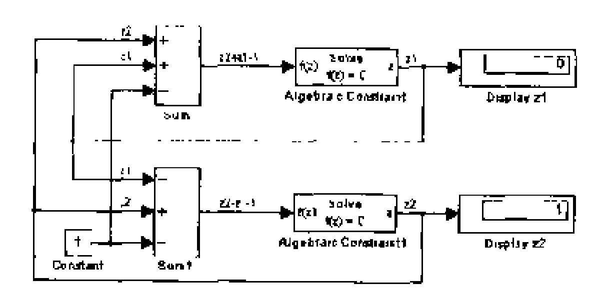


图 6.48　表示（6.12）式描述方程的模型

图 6.48 显示的模型表示（6.12）式描述的方程

$$
\begin{aligned}
z2+z1&=2\\
z2-z1&=2
\end{aligned}
\tag{6.12}
$$

其结果是 z2=1,z1=0，这与模型中的 Display 显示的一致。

数据类型　接受和输出双精度实数信号。

参数　Initial guess：对结果的初始估计值。缺省值为 0。

特点

直通	有
采样时间	由驱动模块继承
标量扩展	无
向量化	可以
零点穿越	无

6.5.3　Bitwise Logical Operator 模块

Bitwise Logical Operator 模块的属性对话框如图 6.49 所示。

该模块可以对输入的无符号整型实现按位逻辑操作，包括封装（AND、OR、XOR）、取反（NOT）和切换操作（SHIFT_LEFT, SHIFT_RIGHT）。也可以对无符号整型的数组实现相同的操作。

（1）封装操作（masking）

Bitwise Logical Operator 模块的封装操作符能够将输入信号与一个称为 mask 的参数按位进行逻辑运算。该参数连同具体的操作符可以在模块的属性对话框中进行设置。无论是输入信号还是 mask 参数本身都支持数组。一般来说，mask 参数必须与输入信号有相同的维数。SIMULINK 将其中的每一个元素与信号的相应元素一一对应。也有例外的情况：

- 如果输入是标量，mask 参数是数组，则该模块输出数组，同时将 mask 的每个元素都与输入信号对应计算。
- 如果输入是数组，mask 参数是标量，该模块输出数组，同时输入的每个元素都与 mask 参数一一对应。
- 如果输入是 1 维数组（即向量），则 mask 参数可以为行向量和列向量。

如果选择封装操作符，可以在属性对话框中的 Second operand 域中输入第一个操作数。用户可以输入任何的标量、向量和元胞数组。如果输入字符串，则表示的是十六进制的数，例如 ‘FFFF’。

必要的时候，该模块会截掉多余的维数来适应输入信号的长度。例如，如果 mask 参数为 ‘FF00’，而输入信号是无符号 8 位整型（uint8），则模块将 mask 参数截断成 ‘00’。

用户也可以使用矩阵来指定十六进制的封装操作，但是需要小心 MATLAB 的书写习惯。例如，表达式[‘FF’　‘00’]表示的是一个字符串 ‘FF00’，而不是两个字符串，同样，[FF’ ; ‘00’]代表两个字符串。但是[‘FF00’ ; ‘00’]则是无效的。可以使用元胞数组来避免类似的错误。例如，图 6.50 用元胞数组（'F0'　'0F'）为 mask 参数定义了十六进制的值，它与两个元素的输入信号相对应。

（2）取反操作

该操作比较简单，它将所有输入信号按位取反后进行输出。

（3）移位操作

移位操作（SHIFT_LEFT 和 SHIFT_RIGHT）可以将输入信号按位移位后输出。可以在属性对话框中的 Second operand 域中指定移动的位数。如果指定的位数大于输入信号本身的位长，则指定移动输入信号本身的位长大小的位数，这时输出位零，mask 参数的维数规则在这里也适用。

数据类型　支持任何无符号类型的实数信号，如 uint8, uint16, uint32 等。输入向量的所有元素都必须有相同的数据类型。输出的数据类型与输入信号相同。

参数　Bitwise operator：指定具体的逻辑操作符。

Second operand：与输入信号进行逻辑操作的第 2 个操作符，根据逻辑操作符的不同，其含义也不同。

特点　直通　有

采样时间　由驱动模块继承

标量扩展　Second operand 参数和输入扩展

状态个数　0

向量化　可以

零点穿越　无

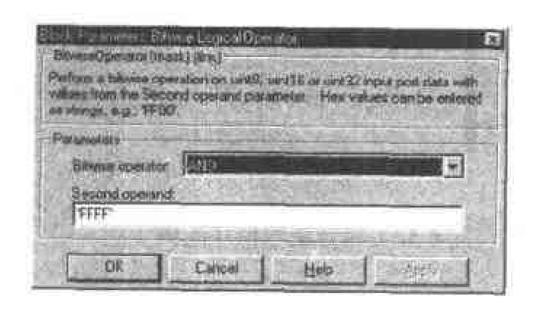


图 6.49　Bitwise Logical Operator 模块的属性对话框

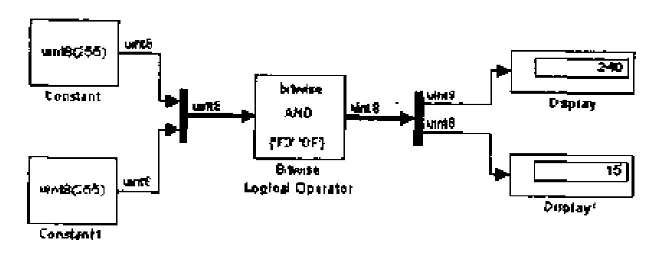


图 6.50　包含 Bitwise Logical Operator 模块的例子

6.5.4　Combinatorial Logic（组合逻辑）模块

Combinatorial Logic 模块的属性对话框如图 6.51 所示。

说明： 该模块能够用在对可编程逻辑阵列（PLAs）、逻辑电路、决策表和其他的逻辑表达式进行建模时，实现一个标准的真值表。

可以指定一个定义模块所有可能的输入矩阵作为 Truth table 参数。矩阵的每一行包含有对应输入元素的不同组合的输出。必须为输入的每一种可能的组合指定一个输出。每个输出都可以是任何数字量，它不局限于取 0 和 1 这两个值。矩阵的列数就使模块的输出数。

模块的输入的个数与矩阵的行数之间的关系是：行数 $=2^{输入数}$。

通过计算一个以输入向量各元素所在的某一行的索引为参数的表达式，SIMULINK 得到要返回的输出矩阵中对应于该行元素的值。SIMULINK 将输入向量为 0 的各元素用 0 代替，不为零的各元素用 1 代替，然后用下面的表达式计算得到行索引，对于一个有 m 个元素的输入向量 u：

$$行索引=1+u(m)\times 2^0+u(m-1)\times 2^1+\cdots+u(1)\times 2^{m-1} \tag{6.13}$$

例如，建立一个有两个输入的与函数的模型。当两个输入的元素都为 1 时返回 1，否则返回 0。要实现这一函数，指定参数 Truth table 的值为[0;0;0;1]。模型中给组合逻辑提供输入和输出的部分如图 6.52 所示。

数据类型　接受逻辑或双精度类型实数信号，输出与输入类型一致。真值表元素也可以是逻辑或双精度类型。如果元素为双精度类型，可以是任何值，不仅仅是 0 或 1 值。如果真值表元素数据类型不同于输出信号的类型，SIMULINK 在计算输出前转换真值表为输出类型。

参数　Turth table：输出矩阵，每一列与输出向量的一个元素相对应，每一行与真值表的一行相对应。

特点　直通　有

采样时间　由驱动模块继承

标量扩展　无

向量化　可以

零点穿越　无

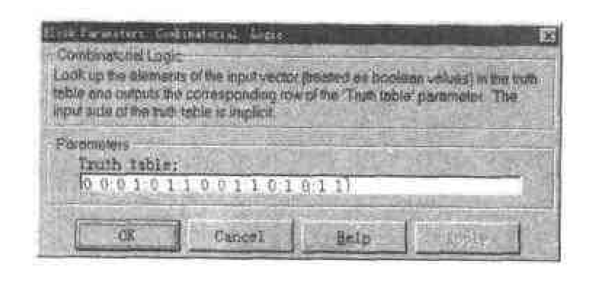


图 6.51　Combinatorial Logic 模块的属性对话框

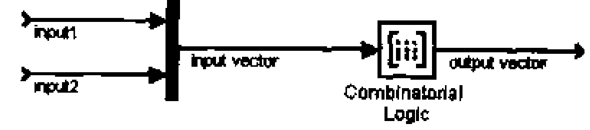


图 6.52　组合逻辑模块的例子

6.5.5　Complex to Magnitude-Angle 模块

Complex to Magnitude-Angle 模块的属性对话框如图 6.53 所示。

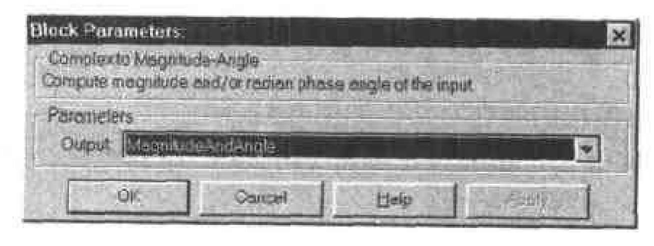


图 6.53　Complex to Magnitude-Angle 模块的属性对话框

说明：该模块接受一复数信号，输出复数信号的幅值和相角。输入可以是复数向量，这时输出也是复数信号向量。幅值向量包含对应复数输入元素的幅值，角度输出包含相应复数输入元素的相角。

数据类型　接受双精度类型复数值信号，输出双精度型的实数。

参数　output：确定模块的输出。从三个值中选取：MagnitudeAndAngle（输出输入信号的幅值和弧度相角，缺省值）、Magnitude（输出输入信号的幅值）、Angle（输出输入信号的弧度相角）。

特点　　直通　　有
　　　　采样时间　由驱动模块继承
　　　　标量扩展　无
　　　　向量化　　无
　　　　零点穿越　无

6.5.6 Complex to Real-Image 模块

Complex to Real-Image 模块的属性对话框如图 6.54 所示。

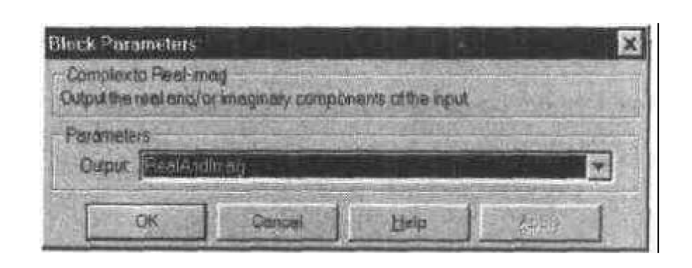


图 6.54 Complex to Real-Image 模块的属性对话框

说明：该模块接受双精度类型的复数信号，输出输入信号的实部和虚部。输入可以是复数信号组成的向量，此时输出也是向量。

数据类型　接受双精度类型复数信号输入，输出双精度类型的实数值。

参数　**Output**：确定模块的输出，可以选择的项包括：RealAndImag（输出输入信号的实部和虚部，缺省值）、Real（输出输入信号的实部）和 Image（输出输入信号的虚部）。

特点　　直通　　有
　　　　采样时间　由驱动模块继承
　　　　标量扩展　无
　　　　向量化　　可以
　　　　零点穿越　无

6.5.7 Dot Product 模块

Dot Product 模块的属性对话框如图 6.55 所示。

说明：该模块产生两个输入向量的点乘，标量输出 y 等于 MATLAB 的运算

y=sum(conj(u1)*u(2))　　　　(6.14)

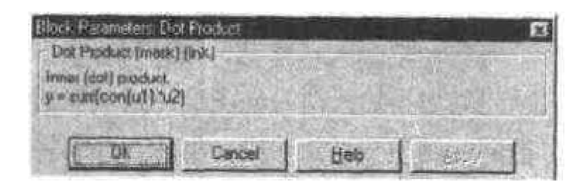


图 6.55 Dot Product 模块的属性对话框

其中，$u1$ 和 $u2$ 表示向量输入，如果两个输入都是向量，它们的长度必须相同。输入向量的元素可以是实数或复数，输出的类型依赖于输入的类型。要执行元素对元素的乘积而不将它们相加，使用 Product 模块。

数据类型	接受和输出双精度类型信号。	
特点	直通	有
	采样时间	由驱动模块继承
	标量扩展	有
	状态个数	0
	向量化	可以
	零点穿越	无

6.5.8　Gain 模块

Gain 模块的属性对话框如图 6.56 所示。

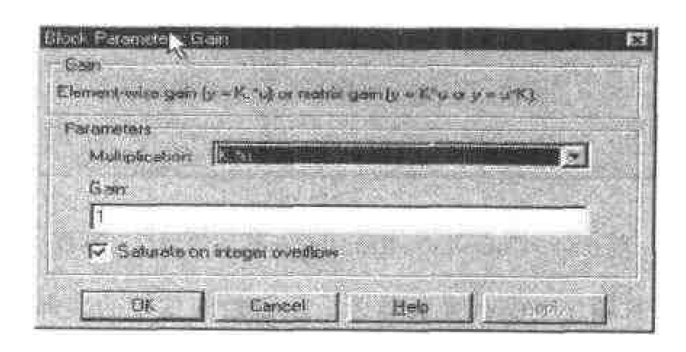


图 6.56　Gain 模块的属性对话框

说明：该模块将其输入乘以一个指定的常数、变量或表达式作为它的输出。

增益可以是数值、变量或表达式。输入或增益可以是标量、向量或矩阵。通过 Multiplication 参数可以指定输入与增益的相乘方式，即是各个元素分别相乘（element-by-element）还是遵循矩阵乘法规则（matrix）。

如果模块的图标足够大，图标上将会自动显示 Gain 参数域中的输入值。如果指定的是变量，图标上也会相应显示该变量的名字。

如果要想在仿真阶段修改增益值，可以使用 Slider Gain 模块。

数据类型　依赖于用户设定的相乘方式。

对于矩阵乘法，输入和增益必须是单精度或双精度的实数或复数形式的标量、向量或矩阵。

对于各个元素分别相乘的方式，输入和增益必须是除布尔类型以外的任何类型的实数或复数形式的标量、向量和矩阵。输出信号的类型与输入信号的类型相同。同时，输入信号的各个元素必须是同一类型。而 Gain 参数也可以是任何类型的实数或复数形式的标量、向量和矩阵。输入输出服从下面的约定：

（1）如果输入是实数，增益是复数，则输出是复数。

（2）如果输入信号的类型与增益参数的类型不同，模块将尽量将增益参数转换成输入信号的同一类型数据。否则，SIMULINK将停止仿真并通知错误。例如，如果输入信号是无符号8位整型（unit8），而增益参数是-1，则会发生错误。如果在类型转换过程中降低了原有数据的精度，则SIMULINK发出警告，并继续仿真过程。

（3）如果输出信号是整型，而Saturate on integer overflow选项被选中，则当数据超过了该数据类型的最大值时，模块进入饱和状态。换句话说，模块将输出其最大值加1或最小值。例

如，如果数据类型定义为int8，则当数据超过127时实际输出为127，而当数据小于-128时，实际输出为-128。

参数　　Multiplication：指定输入与增益相乘的方式：

- K.*u（各个元素进行相乘）
- K*u（将增益作为左操作数的矩阵乘法运算）
- u*K（将增益作为右操作数的矩阵乘法运算）

Gain：增益参数，可以指定为标量、向量、表达式和变量名等，缺省值为1，如果没有特殊声明，其数据类型为双精度型。如果显示的数据太长并且选择了各个元素进行相乘的方式，则在该域中显示-k-字符串。

Saturate on integer overflow：仅仅在选择了各个元素进行相乘的方式时有效。如果该项被选中，则模块在数据溢出时进入饱和状态，否则当数据溢出时，该模块将激活诊断对话框中（Diagnostics）的数据溢出事件。

特点

直通	有
采样时间	由驱动模块继承
标量扩展	输入扩展和 Gain 参数扩展
状态个数	0
向量化	可以
零点穿越	无

6.5.9　Logical Operator 模块

Logical Operator 模块的属性对话框如图6.57所示。

说明：该模块对其输入执行这样一些逻辑运算：AND，OR，NAND，NOR，XOR和NOT，输出取决于输入的数目，它们的向量的大小和选用的操作符如果为TRUE，则输出为1，如果为FALSE，则输出为零。模块的图标显示了所采用的操作符。

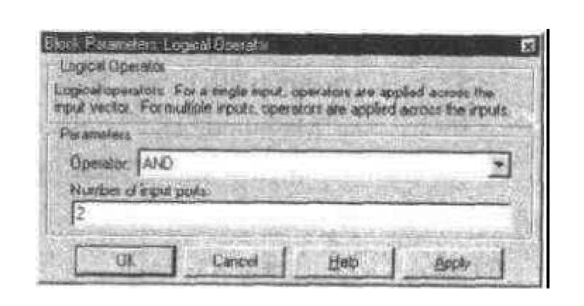


图6.57　Logical Operator 模块的属性对话框

对于两个或多个输入，模块对所有的输入执行逻辑操作。如果输入是向量，将对各个向量的对应元素执行逻辑操作并生成一个输出向量。

对于一个单一的向量输入，模块对向量的所有执行逻辑操作（除NOT外）。NOT操作符只接受一个输入，该输入可以是标量也可以是向量，如果输入是向量，输出向量中元素的个数与输入向量的个数相同，输出向量的元素是输入向量对应元素的非。

数据类型　　接受逻辑类型的信号。除非逻辑兼容模式是激活的，此时可以接受双精度类型输入，非零的输入被看作是TRUE(1)，值为零的输入被看作是FALSE(0)。所有输入必须类型相同，输出与输入类型一致。

参数　　Operator：模块要使用的逻辑运算符，从该参数的选项菜单中选取。

Number of input ports：模块的输入数，其数量必须与所选的运算符相符。

特点	直通	有
	采样时间	由驱动模块继承
	标量扩展	输入扩展
	向量化	可以
	零点穿越	无

6.5.10　Magnitude-Angle to Complex 模块

Magnitude-Angle to Complex 模块的属性对话框如图 6.58 所示。

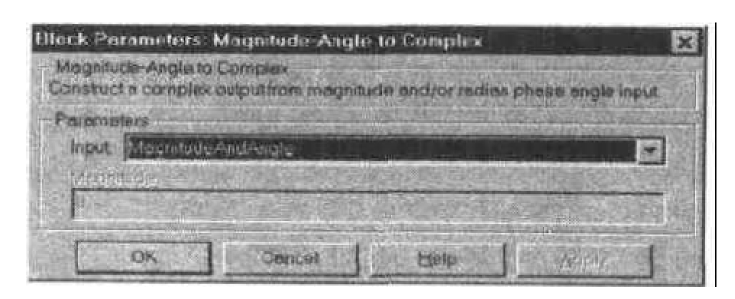


图 6.58　Magnitude-Angle to Complex 模块的属性对话框

说明：该模块将输入的相角幅值转换成复数信号。输入必须是双精度类型实数信号，相角以弧度为单位。输入可以是大小相同的向量，或者是一个输入是一个向量，另一个是标量。幅值输入和相角输入向量分别映射复数输出的相应元素。如果其中一个输入为标量，它将映射输出的所有元素。

数据类型　该模块输入为双精度类型实数信号，输出信号的数据类型为双精度复数。

参数　Input：指定输入种类，包括：Magnitude（幅值输入），Angle（相角输入），MagnitudeAnd Angle（幅值和相角输入，缺省值）。

Angle：如果选择幅值输入，给输出指定一常数弧度相角。缺省值为 0。

Magnitude：如果选择相角输入，给输出指定一常数幅值，缺省值为 0。

特点	直通	有
	采样时间	由驱动模块继承
	标量扩展	输入扩展
	向量化	可以
	零点穿越	无

6.5.11　Math Function 模块

Math Function 模块的属性对话框如图 6.59 所示。

说明：该模块可以执行许多常用的数学函数。这些函数包括：exp、log10u、log10、magnitude2、square、sqrt、pow、conj、reciprocal、hypot、rem、mod、transpose 和 hermitian。该模块的输出是对输入执行指定的函数运算的结果。

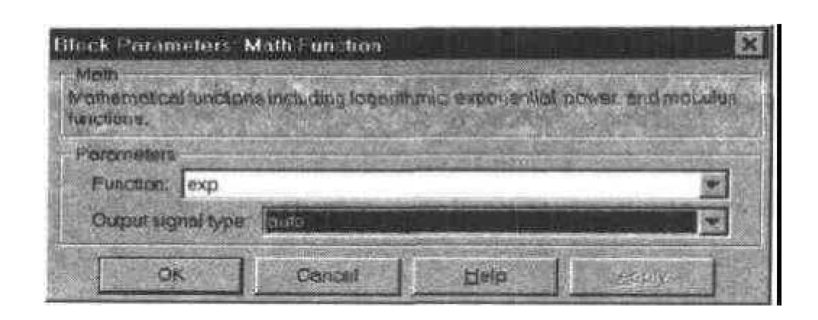


图 6.59　Math Function 模块的属性对话框

函数的名字显示在模块的图标中。SIMULINK自动地画出适当数目的输入端口。

需要输出向量化的输出是应使用Math Function模块而不是Fcn模块。因为Fcn模块只能产生标量的输出。

数据类型　　接受实数和复数值信号或双精度类型信号向量。输出信号类型依据Output signal type参数设定，即实数或复数。

参数　　Function：选择采用的数学函数。

Output signal type：选择数学函数模块输出信号的类型为：实数（real），复数（complex）和自动（auto）。输入与输出数据类型的对应关系可以查阅相应的文档。

特点　　直通　　有

采样时间　　由驱动模块继承

标量扩展　　可以

向量化　　可以

零点穿越　　无

6.5.12　Matrix Gain 模块

Matrix Gain 模块的属性对话框如图 6.60 所示。

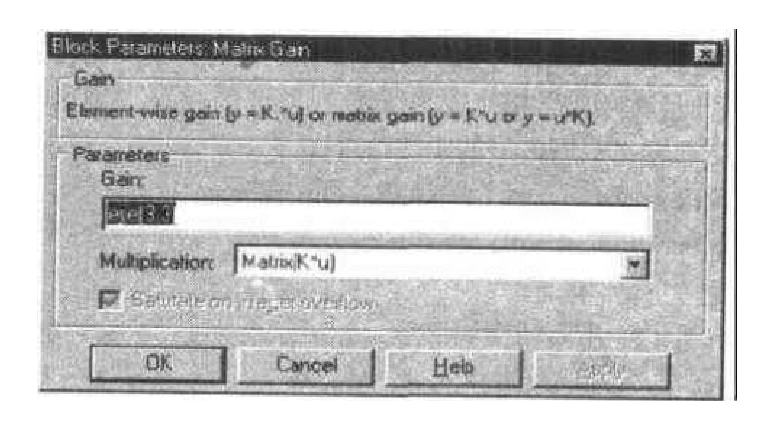


图 6.60　Matrix Gain 模块的属性对话框

说明：该模块实现一个矩阵增益。增益的缺省值为单位矩阵。详细情况可以参考 Gain 模块的说明。

数据类型　　参考 Gain 模块。

参数　　Gain：增益指定为矩阵，缺省值为 eye(3,3)。

Multiplication：输入信号与矩阵增益的相乘类型，缺省值为矩阵乘法。详细情况可以参考 Gain 模块的相关描述。

Saturate on Integer Overflow：在数据溢出时模块是否进入饱和状态。

特点　　直通　　有

采样时间　　连续

标量扩展　　无

状态个数　　0

向量化　　可以

零点穿越　　无

6.5.13　MinMax 模块

MinMax 模块的属性对话框如图 6.61 所示。

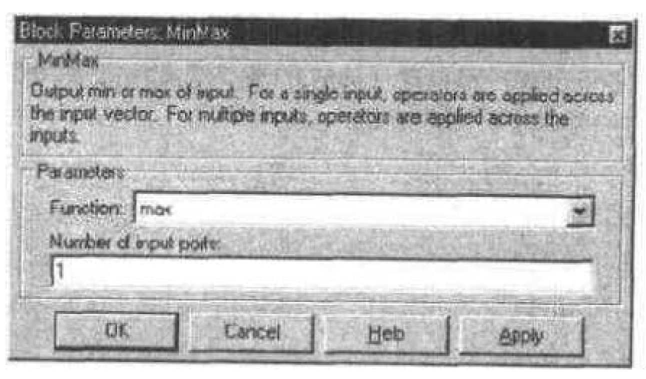


图 6.61　MinMax 模块的属性对话框

说明：该模块输出其输入的最小元素或者最大元素。可以从 Function 参数列表中选择合适的选项，以使用相应的函数：Min 和 Max。

如果模块有一个输入端口，模块的输出是一个标量，它是输入向量的最小或者最大元素。

如果模块有多个端口，模块对各个输入向量进行元素对元素的比较，模块输出向量的每一个元素是各个输入向量对应元素相比较的结果。

数据类型　　接受和输出双精度型的实数信号。

参数　　Function：应用于输入的函数（min 或 max）。

Number of input port：模块的输入数。

特点　　直通　　有

采样时间　　由驱动模块继承

标量扩展　　输入扩展

状态个数　　0

向量化　　可以

零点穿越　　有，检测最小或最大值

6.5.14　Product 模块

Product 模块的属性对话框如图 6.62 所示。

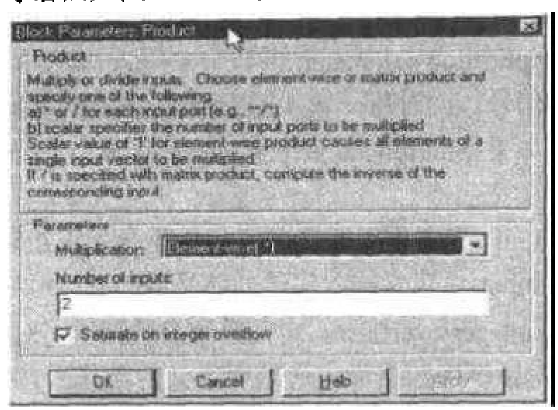


图 6.62　Product 模块的属性对话框

说明：该模块对其输入进行乘或除运算，具体乘除取决于 Multiplication 和 Number of inputs 参数的值。

如果 Number of inputs 参数的值是*和/的组合，并且模块的输入的个数与乘除符号的个数相等，这时模块的图标在每一个输入端口附近显示适当的乘除符号，模块的输出是所有标以“*”的输入的乘积，除以所有标有“/”的输入得到的结果。

如果 Multiplication 参数为 element-wise，则模块按元素将标以“*”的输入相乘，除以所有标有“/”的输入。例如，如果输入向量的大小为 n，则大小为 n 的输出向量的每一个元素等于

$$y_i = u1_i \times u2_i \times \cdots \times un_i \tag{6.15}$$

如果其中一个输入为矩阵，则所有的输入都应该是矩阵或标量，这里的标量定义为 1×1 的矩阵或一个元素的向量。如果一个输入为向量，则其他的输入同样是类向量型，这里的类向量型指标量、向量、行或列矩阵。所有的非标量的输入都具有相同的维数。输入不能同时包含行矩阵和列矩阵。

如果 Multiplication 参数为 matrix，则模块对所有标记“*”的输入使用矩阵乘法，而对标记“/”的输入使用逆矩阵相乘。操作数的顺序在 Number of Inputs 参数域中指定。例如，对于“*/*”的设置，将导致 $AB^{-1}C$ 的运算结果。这里 A,B,C 分别是第 1、2 和 3 的输入信号。输出矩阵的维数遵循矩阵乘法的原则。

如果 Number of inputs 参数指定为“*”，而 Multiplication 参数指定为 element-wise，并且输入是类向量的，即一维数组或二维数组的一行或一列。模块将输出输入向量各元素的相乘解。即

$$y = \prod u_i \tag{6.16}$$

这时图标也将相应变成图 6.63 所示的情况。

如果 Number of inputs 参数的值为“/”，而 Multiplication 参数指定为 element-wise，并且输入是类向量的，则模块输出输入向量各元素的相乘解的逆。如果此时输入一个矩阵，而 Multiplication parameter 参数为

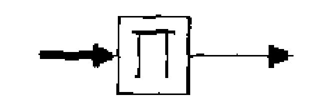


图 6.63　Product 模块的一种图标

element-wise ，SIMULINK 将会出现错误。而如果 Multiplication 参数为 matrix，则模块输出矩阵的逆。

如果模块只有一个输入，它必须是标量或类向量的。如果必要，SIMULINK 会调整图标的大小以显示所有的输入端口。如果输入端口发生改变，则在模块的底部进行增减。

数据类型　如果 Multiplication parameter 参数为 element-wise，则接受任何数据类型的实数或复数信号。同时每一输入的类型都必须相同。输出的数据类型与输入信号相同。如果 Multiplication parameter 参数为 matrix，则输入指定为单精度或双精度类型。

参数　Multiplication：指定采用元素之间相乘还是采用矩阵相乘的乘法规则。

Number of inputs：定义输入信号的序号或“*”和“/”的表达式。缺省值 2。

Saturate on integer overflow：定义模块的数据溢出时是否进入饱和状态，还是激活诊断对话框中数据溢出选项所定义的动作。

特点

直通	有
采样时间	由驱动模块继承
标量扩展	可以
向量化	可以
零点穿越	无

6.5.15　Real-Image to Complex 模块

Real-Image to Complex 模块的属性对话框如图 6.64 所示。

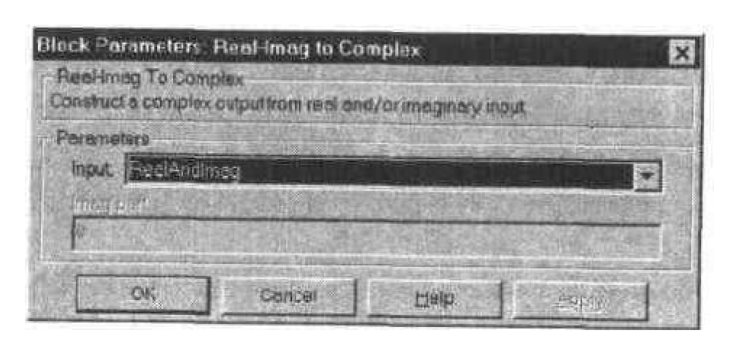


图 6.64　Real-Image to Complex 模块的属性对话框

说明： 该模块将实部和虚部输入转换为复数输出信号。输入可以是大小一样的向量，或者一个输入为向量，另一个为标量。如果输入为向量，则输出为复数信号向量。实部输入向量元素映射相应复数输出元素的实部；虚部输入向量元素同样映射相应复数输出元素的虚部。标量输入映射所有复数输出信号的相应部分（实部或虚部）。

数据类型　输入为双精度类型实数值信号，输出为双精度类型复数值信号。

参数　Input：指定输入种类：一个实部输入（Real），一个虚部输入（Image），或者都输入（RealAndImage）。

Real part：在输入为虚部时，该参数用来指定一个实部常数。

Image part：在输入为实部时，该参数用来指定一个虚部常数。

特点　　直通　　　有
　　　　采样时间　由驱动模块继承
　　　　标量扩展　当需要两个输入时，输入可以标量扩展
　　　　向量化　　可以
　　　　零点穿越　无

6.5.16　Relational Operator 模块

Relational Operator 模块的属性对话框如图 6.65 所示。



图 6.65　Relational Operator 模块的属性对话框

说明： 该模块对两个输入执行关系运算，执行规则为：

- == 　如果第一个输入等于第二个输入时为 TRUE
- != 　如果第一个输入不等于第二个输入时为 TRUE
- < 　如果第一个输入小于第二个输入时为 TRUE
- <= 　如果第一个输入小于或等于第二个输入时为 TRUE
- >= 　如果第一个输入大于或等于第二个输入时为 TRUE
- > 　如果第一个输入大于第二个输入时为 TRUE

如果结果是 TRUE，输出为 1，如果是 FALSE，输出为 0。可以指定输入为标量、向量或者标量与向量的组合。

输入为标量时，输出也是标量；输入是向量时，输出也是向量，并且它的每一个元素是两个输入向量对应元素比较的结果；如果输入是标量与向量的组合，输出是一个向量，它的每一个元素是标量输入与向量输入的对应元素比较的结果。

数据类型　接受任何类型的实数信号。输出为一逻辑类型的信号；在逻辑兼容模式激活时，输出为双精度类型信号。

参数　　Operator：用来选择用于模块输入的关系运算符。

特点　　直通　　　有
　　　　采样时间　由驱动模块继承
　　　　标量扩展　输入可以标量扩展
　　　　向量化　　可以
　　　　零点穿越　无

6.5.17　Rounding Function 模块

Rounding Function 模块的属性对话框如图 6.66 所示。

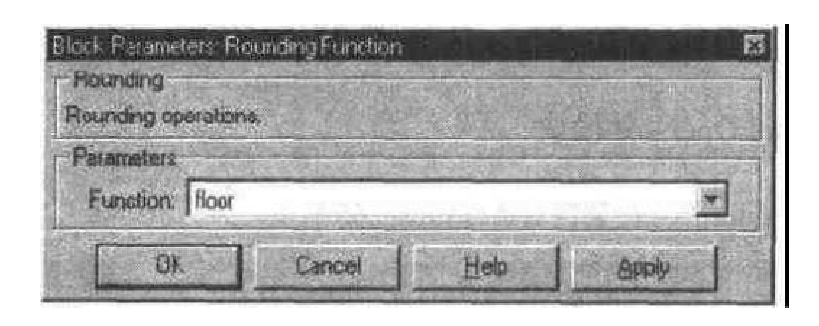


图 6.66　Rounding Function 模块的属性对话框

说明：该模块执行普通的数学圆整函数。

可以从 Function 列表中选择 floor、ceil、round 和 fix 这些函数中的一个。模块的输出是对输入执行相应函数的结果。

函数的名称显示在模块的图标中。

如果想对输出进行向量化，应使用 Rounding Function 模块而不是 Fcn 模块。因为 Fcn 模块只能生成标量输出。

数据类型	该模块接受和输出双精度类型实数信号。	
参数	Function：选择具体的圆整函数。	
特点	直通	有
	采样时间	由驱动模块继承
	标量扩展	不适用
	向量化	可以
	零点穿越	无

6.5.18　Sign 模块

Sign 模块的属性对话框如图 6.67 所示。

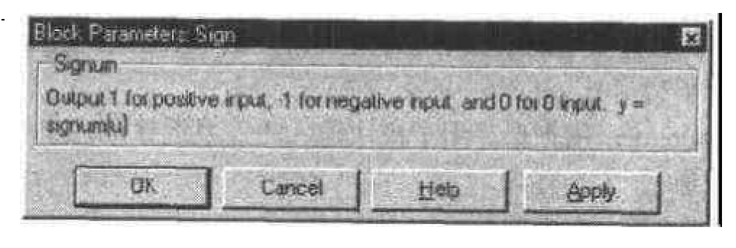


图 6.67　Sign 模块的属性对话框

说明：该模块显示输入信号的符号：

当输入大于 0 时输出为 1；当输入等于 0 时输出为 0；当输入小于 0 时输出为-1。

数据类型　　接受和输出双精度类型实数信号。

特点　　直通　　　有
　　　　采样时间　由驱动模块继承
　　　　标量扩展　不适用
　　　　向量化　　可以
　　　　零点穿越　有，检测何时经过 0 点

6.5.19　Slider Gain 模块

Slider Gain 模块的属性对话框如图 6.68 所示。

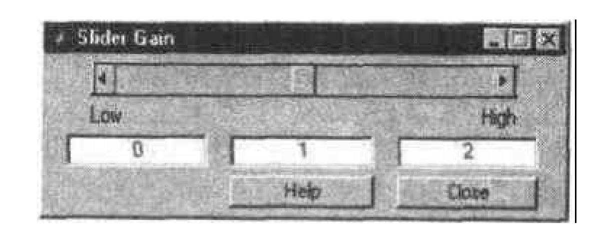


图 6.68　Slider Gain 模块的属性对话框

说明： *该模块可以在仿真过程中使用滑块改变标量增益，该模块接受一个输入并且参数一个输出。*

数据类型　接受除逻辑类型外的任何类型实数或复数值标量、向量，输出与输入类型相同。输入向量元素必须类型相同，增益参数可以是在任何类型的实数或复数值标量。

参数　　Low：滑块范围的下限。缺省值为 0。
　　　　High：滑块范围的上限。缺省值为 2。
　　　　该模块的对话框中的编辑框从左到右分别是下限值、当前值和上限值。可以通过两种方法改变增益，一种是通过拖动滑块，另一种是在对话框中的当前值域中输入新的数值。可以通过改变上下限的值改变增益的范围。单击“Close”按钮关闭对话框。
　　　　如果单击滑块的左箭头或右箭头，当前值减小或者增加滑块范围的 1%，如果单击滑块指针任何一边的条形区域，当前值改变滑块范围的 10%。
　　　　如果要使用向量或矩阵增益可以使用 Gain 模块。

特点　　直通　　　有
　　　　采样时间　由驱动模块继承
　　　　标量扩展　Gain 参数扩展
　　　　状态个数　0
　　　　向量化　　可以
　　　　零点穿越　无

6.5.20　Sum 模块

Sum 模块的属性对话框如图 6.69 所示。

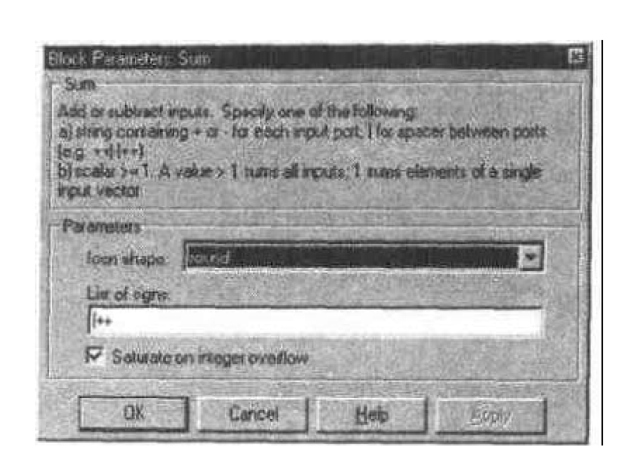


图 6.69　Sum 模块的属性对话框

说明：该模块将各个标量和向量相加，或者当输入只有一个向量时将它的所有元素相加，这取决于输入模块的数目。

如果该模块有多个输入，则所有非标量的输入都必须具有相同的维数。例如，如果其中一个输入是 2×2 的矩阵，则其他的输入也必须是 2×2 的矩阵或者标量。

如果其中某个输入是标量，则它将扩展到与其他输入相同的维数。

如果模块只有一个向量输入，模块的输出是一个标量，它是输入向量各元素之和。

数据类型　接受和输出任何类型的实数或复数值信号。所有输入必须是类型一样。输出与输入类型一样。

参数　Icon shape：从下拉菜单中选择图标的形状，是圆形还是矩形。

List of signs：该参数可以是一个常数或"+"、"-"、"|"符号的组合。指定一个常数表示输入的个数，而"|"符号在相应的位置产生一个空位。

Saturate on integer overflow：如果选中该项，则发生整数溢出时，模块使输出饱和值。

特点

直通	有
采样时间	由驱动模块继承
标量扩展	可以
状态个数	0
向量化	可以
零点穿越	无

6.5.21　Trigonometric Function 模块

Trigonometric Function 模块的属性对话框如图 6.70 所示。

说明：该模块从Function列中选择如下的函数之一：sin、cos、tan、asin、acos、atan、atan2、sinh、cosh和tanh等。该模块的输出是对输入执行函数的结果。

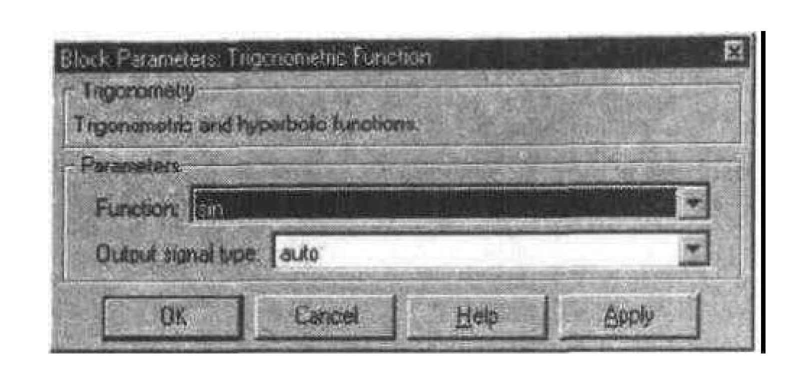


图 6.70 Trigonometric Function 模块的属性对话框

函数的名称显示在模块的图标中。SIMULINK 自动绘出适当数目的输入端口。

如果想得到向量化的输出应使用 Trigonometric Function 模块，而不是 Fcn 模块。后者只能产生标量。

数据类型	接受和输出双精度类型实数或复数信号。	
参数	Function：选择要执行的三角函数。	
	Output signal type：模块。	
特点	直通	有
	采样时间	由驱动模块继承
	标量扩展	当需要两个输入时，输入可以标量扩展
	向量化	可以
	零点穿越	无

6.6 Nonlinear 库

6.6.1 Backlash 模块

Backlash 模块的属性对话框如图 6.71 所示。

说明：该模块可以实现以原点为中心的回滞功能。缺省回滞宽度（又称死带）为 1，并且初始输出为 0 的模块的初始状态如图 6.72 所示。

一个有间隙的系统可以是下面的三种方式之一：

（1）未咬和。在这种方式下，输入不驱动输出，输出保持常数。

（2）正向咬和。在这种方式下，输入是递增的，输出等于输入减去死带宽度的一半。

（3）负向咬和。在这种方式下，输入是递减的，并且输出等于输入加上死带宽度的一半。

如果初始输入在死区的范围之外，初始输出参数值将根据模块在仿真开始时是正向还是负向来确定其输入加上回滞宽度的一半还是减去死带宽度的一半。

例如，Backlash 模块可以用于模拟两个齿轮的耦合。输入和输出都是在某端有一个齿轮的轴，并且输出轴由输入轴驱动。齿轮之间多余的空间引入了间隙，空间的宽度就是参数死带宽度 Deadband width 的值。如果系统初始时未咬和，输出（被驱动齿轮的位置）由初始输

出参数 Initial output 确定。

数据类型	接受和输出双精度类型的实数信号。	
参数	Initial output：初始的输出值，缺省值为 0。	
	Deadband width：死带的宽度，缺省值为 1。	
特点	直通	有
	采样时间	由驱动模块继承
	标量扩展	可以
	向量化	可以
	零点穿越	有，用上下阈值检测接触咬和

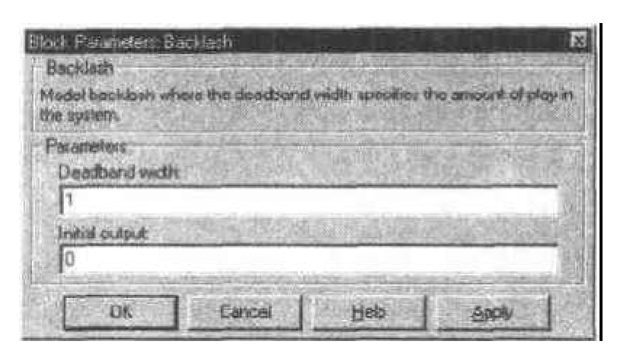


图 6.71　Backlash 模块的属性对话框

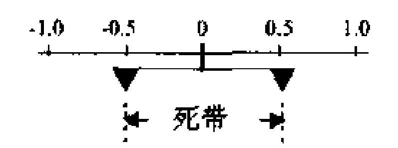


图 6.72　Backlash 模块的回滞示意图

6.6.2　Coulomb & Viscous Friction 模块

Coulomb & Viscous Friction 模块的属性对话框如图 6.73 所示。

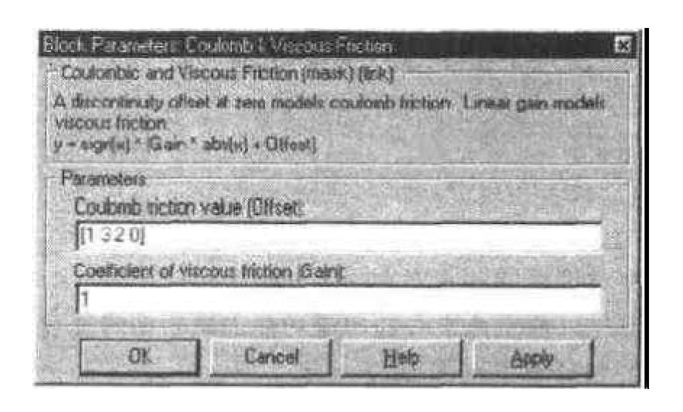


图 6.73　Coulomb & Viscous Friction 模块的属性对话框

说明：该模块能够对库仑（Coulomb）静态摩擦和粘性（Viscous）动态摩擦进行建模。该模块模拟在 0 处不连续，在其他的地方有线性增益的和函数。偏移量（Offset）对应于库仑摩擦，增益（Gain）对应于粘性摩擦。实现模块的表达式为

$$y = sign(u)*(Gain*abs(u)+Offset) \tag{6.17}$$

其中，y 是输出，u 是输入，Gain 和 Offset 是模块的参数。

模块有一个输入和一个输出。

数据类型　接受和输出双精度型实数信号。

参数　　Coulomb friction value（Offset）：对于所有输入值的偏移量。缺省值为[1 3 2 0]。
　　　　Coefficient of viscous friction（Gain）：在非 0 输入点的信号增益。缺省值为 1。

特点　　直通　　　有
　　　　采样时间　由驱动模块继承
　　　　标量扩展　无
　　　　向量化　　可以
　　　　零点穿越　有，克服静态摩擦的点

6.6.3　Dead Zone 模块

Dead Zone 模块的属性对话框如图 6.74 所示。

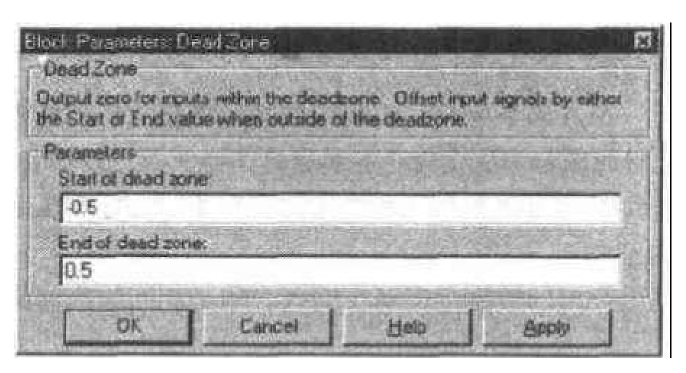


图 6.74　Dead Zone 模块的属性对话框

说明：该模块在指定的区域内产生 0 输出，这一区域也称为死区。死区的下限和上限由参数死区开始（Start of dead zone）和死区结束（End of dead zone）指定。模块的输出取决于输入和死区。

- 如果输出在死区内，则输出为零。
- 如果输入大于或者等于上限，输出是输入减去上限。
- 如果输入小于或者等于下限，输出是输入减去下限。

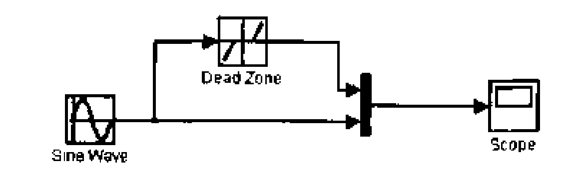


图 6.75　含有 Dead Zone 模块的模型

例如，假设一模型的输入为正弦波，死区的上下限分别为 0.2 和-0.6，模型如图 6.75 所示。

图 6.76 显示了 Dead Zone 模块对正弦波的影响，当输入在-0.6 和 0.2 之间时输出为零。

数据类型　　接受和输出双精度类型的实数信号。

参数　　Start of dead zone：死区下限。缺省值为-0.5。
　　　　End of dead zone：死区上限。缺省值为 0.5。

特点　　直通　　　有
　　　　采样时间　由驱动模块继承
　　　　标量扩展　参数扩展
　　　　向量化　　可以
　　　　零点穿越　有，检测死区上下限

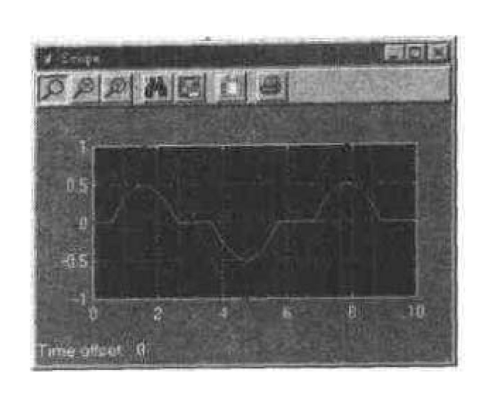


图 6.76　模型的输出曲线

6.6.4　Manual Switch 模块

说明：该模块实现一个在两个输入之间进行手动切换功能的切换开关。要在两个输入之间进行拨动，只要双击它的图标（该模块没有属性对话框）。被选取的输入传递给输出，而没有被选取的输入被舍去。可以在仿真开始之前设置好开关，当仿真在运行的时候，可以切换开关以控制信号的流向。

数据类型　接受任何类型的数据，但两个输入必须是相同的数值或数据类型。输出与输入类型一致。

特点	直通	有
	采样时间	由驱动模块继承
	标量扩展	不适用
	向量化	可以
	零点穿越	无

6.6.5　Multiport Switch 模块

Multiport Switch 模块的属性对话框如图 6.77 所示。

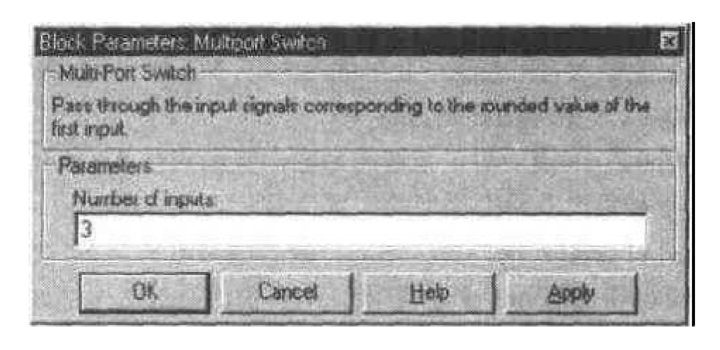


图 6.77　Multiport Switch 模块的属性对话框

说明：该模块从模块的多个输入当中选择一个作为输出。

第一个输入是控制信号，其他的输入是数据输入端。控制端的值确定了哪一个输入数据传送给输出。SIMULINK 将控制端输入值圆整为最为接近的正整数，并且切换到与这一数相

对应的数据输入端。

- 小于 1 或大于数据输入端口的数目：输出越界错误。
- 接近于 1 ：输出第一个数据输入端的信号。
- 接近于 2 ：输出第二个数据输入端的信号。
- 以下类推

数据输入端可以是标量也可以是向量，同样，控制输入端可以是标量也可以是向量，模块的输出遵循下面的规则：

- 如果输入是标量，输出也是标量。
- 如果模块有多个数据输入端口，并且只有一个输入是向量，任何标量输出被扩展为向量。
- 如果模块只有一个数据输入端口，并且该输入端是向量，模块的输出是向量中对应于控制输入端圆整值的元素。

数据类型　　控制输入接受除逻辑类型以外的任何内建数据类型实数信号。数据输入端接受任何类型的实数或复数值输入。所有数据输入必须是类型一样的，输出与输入数据类型一样。

参数　　Number of inputs：模块的数据输入个数。

特点　　直通　　有
　　采样时间　　由驱动模块继承
　　标量扩展　　可以
　　向量化　　可以
　　零点穿越　　无

6.6.6　Quantizer 模块

Quantizer 模块的属性对话框如图 6.78 所示。

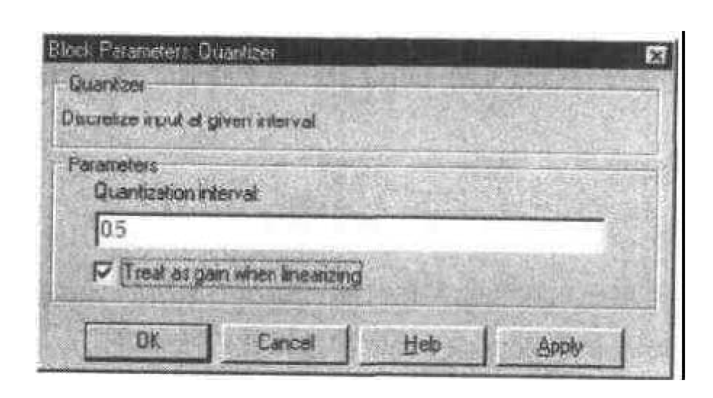


图 6.78　Quantizer 模块的属性话框

说明：该模块将其输入信号传给阶梯函数，使得输入轴上连续的一段区间映射为输出轴上的一点。它相当于将一个光滑的信号量化成阶跃的输出。输出是通过使用圆整为最邻近的点的方法得到的。

该模块产生的结果是关于零点对称的：

$$y = q * round(u/q) \qquad (6.18)$$

其中，u 是输入，q 是 Quantization interval 参数。

数据类型　接受和输出单精度或双精度的实数和复数信号。

参数　　Quantization interval：输出被量化的间隔，允许输出值为 n*q，n 为整数，q 为量化间隔。缺省值为 0.5。

Treatas gain when linearizing：如果选中该项（缺省情况），则 SIMULINK 在线性化模型中将该模块简化为单位增益参数的增益模块，这主要是针对大信号而言。否则，SIMULINK 将增益参数置为 0，这主要是针对小信号而言。

特点　　直通　　有

采样时间　由驱动模块继承

标量扩展　参数扩展

向量化　　可以

零点穿越　无

6.6.7 Rate Limiter 模块

Rate Limiter 模块的属性对话框如图 6.79 所示。

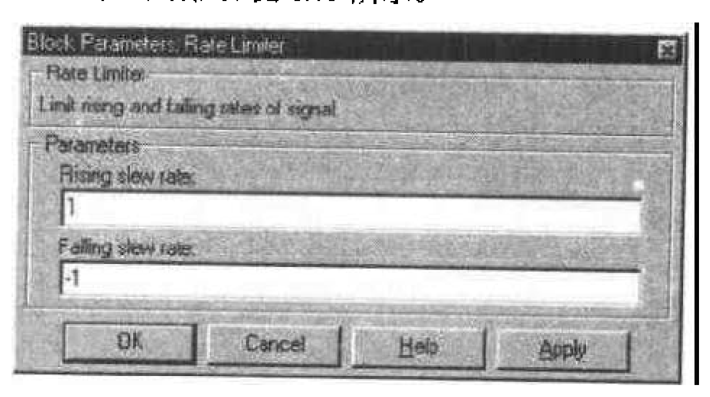


图 6.79　Rate Limiter 模块的属性对话框

说明：该模块限制通过它的信号的一阶导数。换句话说，模块的输出不能比给定的斜率改变的更快。

导数用（6.19）式计算：

$$\text{rate} = \frac{u(i) - y(i-1)}{t(i) - t(i-1)} \tag{6.19}$$

其中，$u(i)$ 和 $t(i)$ 是模块当前的输入和时间。$y(i-1)$ 和 $t(i-1)$ 是前一步的输出和时间。输出由 rate 与 Rising slew 和 Falling slew rate 参数比较的结果确定：

（1）如果 rate 比 Rising slew rate 参数（R）大，输出为：

$$y(i) = \Delta t \cdot R + y(i-1) \tag{6.20}$$

（2）如果 rate 比 Falling slew rate 参数（F）小，输出为：

$$Y(i)=\Delta t \cdot F+y(i-1)$$

（3）如果 rate 在 Rising slew rate 和 Falling slew rate 之间，输出的变化等于输入的变化。

$$Y(i)=u(i) \tag{6.21}$$

数据类型　接受和输出双精度类型的数据。

参数　　Rising slew rate：增长的输入信号导数限，缺省值为 1。

Falling slew rate：减小的输入信号导数限，缺省值为-1。

特点　直通　有
　　采样时间　由驱动模块继承
　　标量扩展　输入和参数可以扩展
　　向量化　可以
　　零点穿越　无

6.6.8　Relay 模块

Relay 模块的属性对话框如图 6.80 所示。

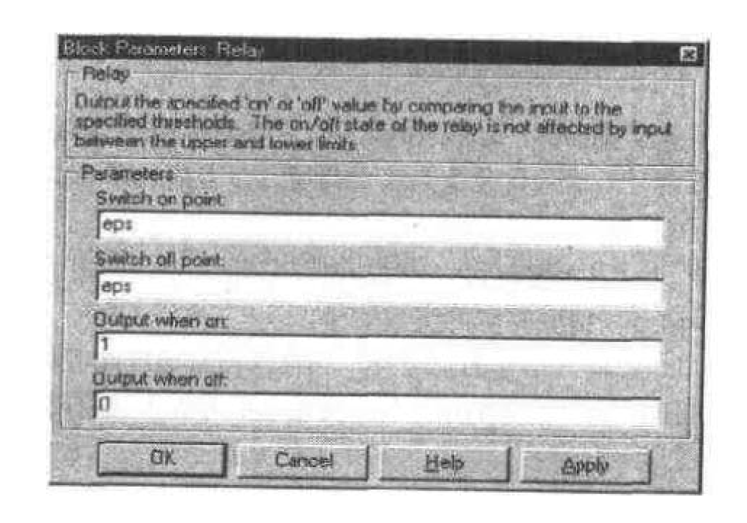


图 6.80　Relay 模块的属性对话框

说明：该模块允许输出在两个给定的值之间切换。当继电器开时，它保持为开的状态，直到输入降得比切断点（Switch off point）参数的值低时为止。当继电器关时，它保持为关的状态，直到输入超过接通点（Switch on point）参数的值时为止。模块接受一个输入并且产生一个输出。

当指定的接通点（Switch on point）值比切断点（Switch off point）值大时，它可以模拟滞后现象；当指定的接通点（Switch on point）值与切断点（Switch off point）值相等时，模拟该值为门槛的开关。接通点的值必须大于或者等于切断点的值。

数据类型　接受和输出双精度类型的实数信号。

参数　Switch on point：继电器接通阈值。缺省值为 eps。
　　Switch off point：继电器断开阈值。缺省值为 eps。
　　Output when on：继电器接通时的输出。缺省值为 1。
　　Output when off：继电器断开时的输出。缺省值为 0。

特点　直通　有
　　采样时间　由驱动模块继承
　　标量扩展　可以
　　向量化　可以
　　零点穿越　有，检测断开和接通点

6.6.9　Saturation 模块

Saturation 模块的属性对话框如图 6.81 所示。

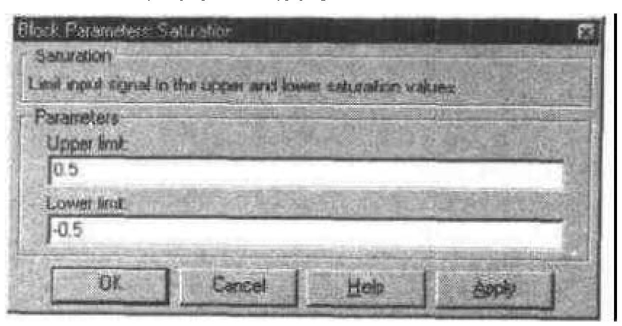


图 6.81　Saturation 模块的属性对话框

说明：该模块对信号设置上、下边界。当输入信号在由下限（Lower limit）和上限（Upper limit）参数指定的范围内时，输入信号毫无改变的通过。当输入信号在边界之外时，信号被减去上边界值或下边界值。

当该模块的 Lower limit 和 Upper limit 参数的值相同时，模块输出与参数值相同的参值。

数据类型　接受和输出双精度类型的实数信号。

参数　　Upper limit：输入信号的上边界，当输入信号超过此值时，模拟输出为该值。

　　　　Lower limit：输入信号的下边界，当输入信号低于此值时，模块的输出为该值。

特点　　直通　　　有

　　　　采样时间　由驱动模块继承

　　　　标量扩展　参数和输入可以标量扩展

　　　　向量化　　可以

　　　　零点穿越　有，检测何时信号到达极限，何时离开极限

6.6.10　Switch 模块

Switch 模块的属性对话框如图 6.82 所示。

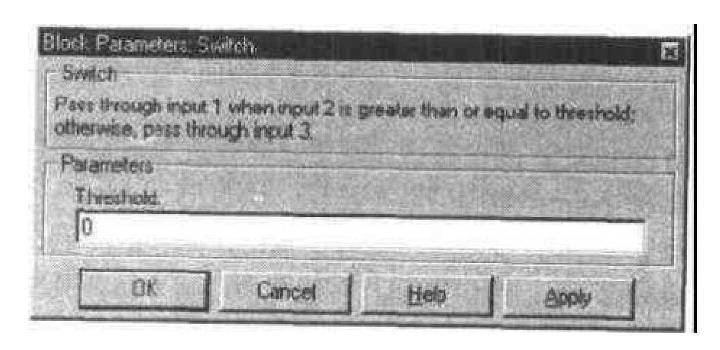


图 6.82　Switch 模块的属性对话框

说明：该模块包含三个输入端，第二个输入为控制输入端口。根据控制输入的值将它的两个输入中的一个传送给输出。如果控制信号（第二个输入端口）大于或者等于阈值

（Threshold）参数值，模块传送第一个输入，否则传送第三个输入。

如果要用逻辑输入（0 或 1）驱动 Switch 模块，将其 Threshold 设为 0。

数据类型　接受任何类型的实数或复数信号作为开关输入（第一、第三端口）。所有输入必须是类型相同的。输出与输入信号的数据类型相同。

参数　Threshold：阈值参数，开关转换的控制（第二个端口）值。该参数值可以指定为标量或与输入向量宽度一样的向量。

特点　直通　有

采样时间　由驱动模块继承

标量扩展　可以

向量化　可以

零点穿越　有，检测开关转换何时发生

6.7 Function & Table 库

6.7.1 Direct Look-Up Table（n-D）模块

Direct Look-Up Table（n-D）模块的属性对话框如图 6.83 所示。

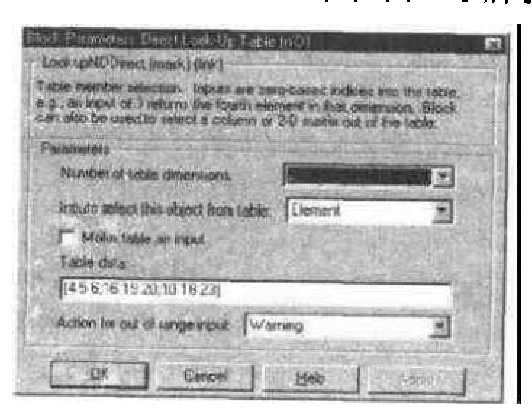


图 6.83　Direct Look-Up Table（n-D）模块的属性对话框

说明：该模块将其输入作为基于零的索引值来进行 n 维查询表的检索。

输入变量的数目根据输出的形状不同而发生变化。输出可以是标量、向量或 2 维矩阵。由于该模块使用的是从零开始的索引输入，因此，整数的数据类型就可以完全描述输入的范围。例如，一个 8 位无符号整型的数据类型就可以完全表示 256 个不同的索引值。

可以在 Table data 参数中定义输入值，也可以指定输出的类型，如标量、向量或矩阵等等。第一个输入表示比输出维数高 1 维的下标值；第二个输入表示比输出维数高 2 维的下标值，以次类推。图 6.84 表示的是输出形状为 2 维矩阵的 5 维查询表，其输出是分别由 R 和 C 定义的行向量和列向量组成。

根据定义的查询表的类型（维数、输出状态等）不同，该模块显示的图标也不同。

例如如图 6.85 所示的模型。Direct Look-Up Table （n-D）模块定义了 4 维查询表，其参

数定义为：

Invalid input value: "Clip and Warn"

Output shape: "Vector"

Table data: int16(a)

这里，a 是由 MATLAB 计算得到的线性增长的 4 维数组：

```
>>a = ones(20, 4, 5, 7); L = prod(size(a));
>>a(1:L) = [1:L]';
```

注意，该模块使用索引是从 0 开始的，因此，该图中的模型输出的是查询表中第二维的第 2 个元素，第三维的第 4 个元素，第四维的第三个元素所对应的第一维的向量，其中包含 20 个元素。



图 6.84　5 维查询表的表示

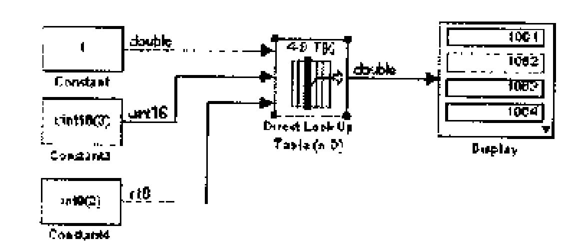


图 6.85　包含 Direct Look-Up Table （n-D）模块的模型

数据类型　可以接受混合类型的输入信号，包括 double、single、int8、uint8、int16、uint16、int32 和 uint32。输出根据输入信号类型的不同而可以是任何的数据类型。如果在模块的一个输入端输入查询表，则输出端口的数据类型是由表的输入端类型继承。索引值输入必须是实数，表中的数据可以是复数。

参数　Number of table dimensions：查询表维数的大小，它可以指定查询表中独立下标的个数。

Inputs select this object from table：指定输出数据的形状。

Make table an input：选中该项后，该模块忽略 Table data 参数的内容，而相应增加一个标示为“T”的输入端口，可以通过该端口直接输入查询表的内容。

Table data：指定查询表的内容。其中查询表的维数必须满足输入输出维数的约定。

Action for out of range input：索引值超过查询表是模块的动作，例如，警告（Warning），错误（Error）或什么也不作（None）。

特点

直通	有
采样时间	由驱动模块继承
标量扩展	仅对于标量查询存在（返回列向量或矩阵的情况除外）
向量化	仅对于标量查询存在（返回列向量或矩阵的情况除外）
零点穿越	无

6.7.2 Fcn 模块

Fcn 模块的属性对话框如图 6.86 所示。

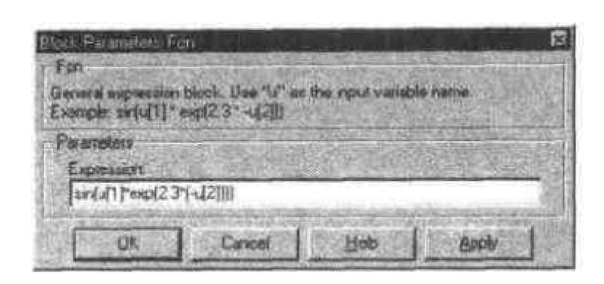


图 6.86 Fcn 模块的属性对话框

说明：该模块对其输入使用指定的 C 语言风格描述的表达式。

表达式可以用一或多个如下元素组成：

u—模块的输入。如果 u 是一个向量，u(i)表示向量的第 I 个元素；u(1)或者 u 表示第一个元素；

- 数字常量；
- 算术运算符（+，-，*，/）；
- 关系运算符（= =，! =, >, <, > =, <=），如果表达式运算结果为真时返回 1；否则返回 0；
- 逻辑运算符（&&, ||, !），如果表达式运算结果为真时返回 1；否则返回 0；
- 圆括号；
- 数学函数：abs, acos, asin, atan, atan2, ceil, cos, cosh, exp；
- fabs, floor, hypot, ln, log, log10, pow, power, rem, sgn, sin, sinh, sqrt, tan,
- 和 tanh；
- 工作空间变量，不能被识别为上面所列出的各项的变量名将传给 MATLAB 求值。矩阵和向量的元素必须被指定（例如矩阵的第一个元素用 A(1, 1)而不是 A）。

优先规则遵循 C 语言的标准。

（1）()

（2）+，-（一元）

（3）pow（幂）

（4）!

（5）*，/

（6）+，-

（7）>, <, <=, >=

（8）= =, ! =

（9）&&

（10）||

这里的表达式有别于 MATLAB 表达式的一个地方是它不能执行矩阵运算。另外，该模块也不支持冒号运算。

模块的输入可以是标量也可以是向量。输出通常是标量。对于向量输出，考虑使用 Math Function 模块。如果模块是向量，函数运算对输入的各个元素分别进行运算（如 sin 函数），模块只对元素的第一个元素进行运算。

数据类型　接受和输出双精度类型的信号。

参数　Expression：对输入要执行的 C 语言规则的表达式。

特点　直通　有

采样时间　由驱动模块继承

标量扩展　无

向量化　无

零点穿越　无

6.7.3 Look-Up Table 模块

Look-Up Table 模块的属性对话框如图 6.87 所示。

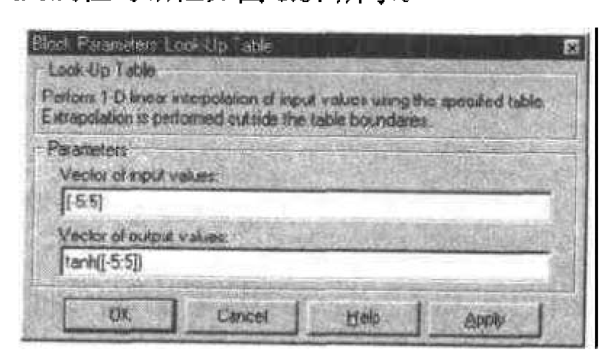


图 6.87　Look-Up Table 模块的属性对话框

说明：该模块使用模块参数中定义的线性插值将输入映射到输出。可以指定（用行向量或者列向量）定义查询表的输入值向量（Vector of input values）和输出值向量（Vector of output values）参数。模块将其输入与输出值向量中的值进行比较产生输出。

如果在输入值向量参数中找到了模块的输入值，则输出是输出值向量参数中相应的元素。

如果在输入向量参数中没有找到模块的输入值，则在表中相邻两个适当的元素之间进行线性插值以确定输出的值。如果实际输入比输入向量参数表中第一个元素还要小或者比最后一个元素还要大，模块使用开始的两个或最后的两个元素进行外推。

如果需要将两个输入映射到一个输出，可以使用 Look-Up Table（2-D）模块。

要建立一个阶跃的查询表，在重复输入参数的值时该变输出值。

当对于一个给定的输入值有两个点时，模块根据如下的原则产生输出：

当 u 小于 0 时，取与从原点沿负轴方向遇到的第一个点相对应的值作为输出。

当 u 大于 0 时，取与从原点沿正轴方向遇到的第一个点相对应的值作为输出。

当 u 在原点并且与零输入对应的有两个输出值，实际的输出是它们的平均值。如果对应于零输入有三个输出值，模块产生的输出值是中间的一个。

Look-Up Table 模块的图标显示了输入向量相对于输出向量的图形。改变模块对话框中的某一个参数时，单击“Apply”或者“OK”按钮时图标自动地重新绘制。

数据类型　接受和输出双精度类型的信号。

参数	Vector of iuput value：输入值向量，包含可能的模块的输入值。该向量与输出向量必须大小一致，输入向量必须单调增加。 Vector of output value：包含模块的输出值的向量。与输入向量必须大小一致。	
特点	直通	有
	采样时间	由驱动模块继承
	标量扩展	无
	向量化	可以
	零点穿越	无

6.7.4　Look-Up Table（2-D）模块

Look-Up Table（2-D）模块的属性对话框如图 6.88 所示。

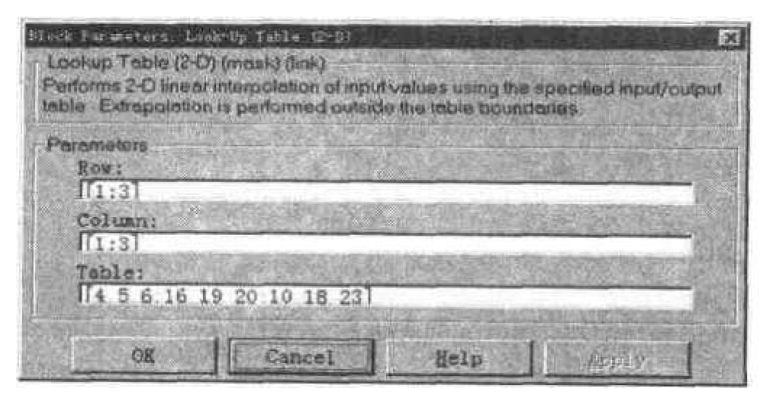


图 6.88　Look-Up Table（2-D）模块的属性对话框

说明：该模块使用模块参数中定义的线性插值将输入映射到输出。

可以在 Table 参数域中定义可能的输出，在行和列参数域中定义与表的行和列对应的值。模块将其输入与行和列参数进行比较，产生其输出值。第一个输入被认为是行的值，第二个输入被认为是列的值。

模块根据其输入值产生输出。

如果输入匹配了行和列参数的值，输出就是表中行和列相交点的值。

如果输入没有匹配行和列参数的值，模块在表中合适的值之间进行线性插值以产生输出。如果模块至少有一个输入比行或者列参数值的第一个还要小或者比最后一个还要大，模块就根据开始或者最后两个点进行外推。

如果行或者列参数之一有重复的值，模块使用 Look-Up Table 模块中介绍的方法选择相应的值。

例如，模块的参数定义为：

Row: [1 2]

Column: [3 4]

Table: [10 20; 30 40]

图 6.89 显示了定义的二维查询表和包含该查询表的一个例子。这里显然输出的是行参数（1）和列参数（4）相交点的值（20）。

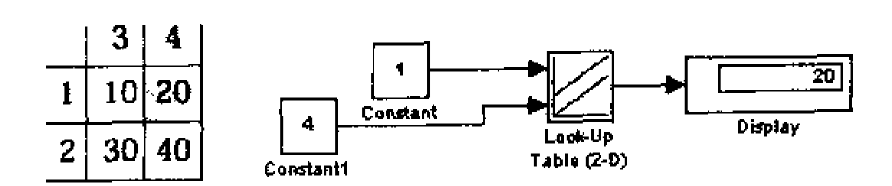


图 6.89　包含 Look-Up Table（2-D）模块的例子

数据类型　接受和输出双精度类型的信号。

参数　Row：表的行值，以向量形式输入，必须单调增加。

Column：表的列值，以向量形式输入，必须单调增加。

Table：输出值表，矩阵大小必须与行（Row）列（Column）参数定义的维数相匹配。

特点

直通	有
采样时间	由驱动模块继承
标量扩展	有
向量化	可以
零点穿越	无

6.7.5　Look-Up Table（n-D）模块

Look-Up Table（n-D）模块的属性对话框如图 6.90 所示。

图 6.90　Look-Up Table(n-D)模块的属性对话框

说明：该模块通过线性插值给出含 n 个变量的近似函数 $y = F(x1, x2, \cdots, xn)$ 的输出，它通过模块中定义的查询表的线性插值将模块的输入映射到输出。插值的方法包括：

- Flat（常数插值）
- Linear（线性插值）
- Natural（立方插值）

这些方法适用于 1 维、2 维和更高维的查询表。

在 Table data 参数域中定义输出数据的内容。模块通过输入与查询表的比较来确定输出。第 1 个输入对应于第 1 维的下标，第 2 个输入对应于第 2 维的下标，以此类推。

如果模块在查询表中找到了相应的下标索引，则直接输出索引对应的数据，否则，模块通过相邻数据的插值来计算输出数据。

如果要使用非插值的查询表，可以采用 Direct Look-Up Table（n-D）模块。

数据类型　该模块可以接受单精度或双精度的数据，但是对于任何给定的 n 维查询表，其输入必须是同类型的。输出类型与输入的数据类型相同。

参数　Number of table dimensions：Table data参数值的维数，它决定了查询表中独立

下标的个数，因此也决定了输入的个数。

First input （row） breakpoint set：指定表中定义的行值，输入为单调增长的向量形式。

Second （column） input breakpoint set：指定表中定义的列值，输入为单调增长的向量形式。该参数域只在表的维数大于1时有效。

Third ... Nth input breakpoint set：指定表中对应于第三维的值，输入为单调增长的向量形式。该参数域只在表的维数大于2时有效。

Fourth input breakpoint set：指定表中对应于第四维的值，输入为单调增长的向量形式。该参数域只在表的维数大于3时有效。

Fifth..Nth input breakpoint sets (cell array)：指定表中对应于第3，4和更高维的元胞数组，输入1维元胞数组向量形式。例如，包含两个向量的元胞数组{[10:10:30], [0:10:100]}将被用于查询表中第5和第6维的数据。向量的值必须单调增长。

Explicit number of dimensions：当表的维数超过4时的表的维数，当用户定义了更高维的表时该参数才有效。

Index search method：查询表索引搜索的方式。选择“Evenly Spaced Points”（间隔搜索），“Linear Search”（线性搜索）或“Binary Search”（按位搜索，缺省值）之一。不同的搜索方式其速度也不同。

Begin index searches using previous index results：当选中该项后，模块将使用前一时刻的索引值来初始化索引搜索。如果相邻仿真步之间输入信号在查询表中的位置变化不大时，选中该参数项能够很大程度上加快查询的速度。

Use one （vector） input port instead of N ports：指定模块是否采用单个的输入来替代一个独立变量对应与一个输入端的方式。这在定义很大的查询表时可以简化模块方框图的显示。

Table data：定义查询表的具体内容，其维数必须满足相关的规则。

Interpolation method：选择某种插值方法。

Extrapolation method：选择某种外推方法。

Action for out of range input：索引超出范围时采取的动作。包括警告（Warning），错误（Error）或什么也不作（None）。

特点		
	直通	有
	采样时间	由驱动模块继承
	标量扩展	无
	向量化	无
	零点穿越	无

6.7.6 MATLAB Fcn 模块

MATLAB Fcn 模块的属性对话框如图 6.91 所示。

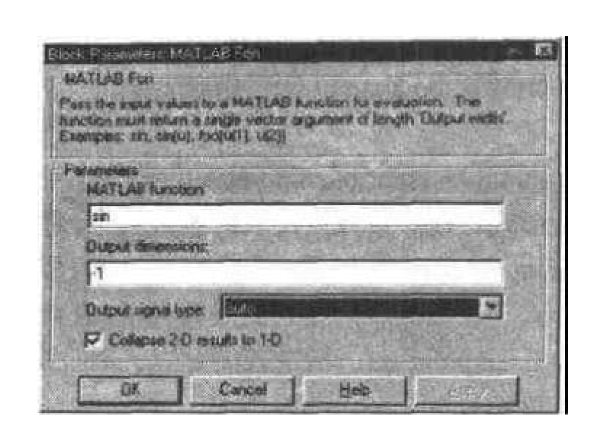


图6.91　MATLAB Fcn模块的属性对话框

说明：该模块对其输入应用指定的 MATLAB 函数或表达式。模块接受一个输入并且产生一个输出。指定的函数或者表达式被用于输入。函数的输出数目必须与模块的输出数目一致，否则会出现错误。

下面是对于该模块来说一些有效的表达式的例子：

sin

atan2(u(1), u(2))

u(1)^u(2)

该模块的执行速度比 Fcn 模块要慢一些，因为它的每一积分步都调用 MATLAB 的解释器。可以考虑采用内建模块（如 Fcn 模块或者 Math Function 模块）来替代它，或者将该函数写成 M 文件形式或者 MEX 文件形式的 S 函数，然后使用 S-Function 模块访问。

数据类型　接受双精度型的实数或复数信号，并产生双精度类型的实数或复数输出。但依赖于输出信号类型（Output signal type）的参数。

参数　MATLAB function：指定要执行的函数或表达式，如果只指定为函数，就不需要包含输入参数。

Output dimensions：输出数据的维数。如果输出的维数与输入相同，则设置为-1，否则，必须指定正确的维数，否则将会发生错误。

Output signal type：输出数据的类型。可以是 real、complex 或 auto 类型之一。如果选择自动（auto），则输出类型与输入类型相同。

Collapse 2-D results to 1-D：将 2 维数组作为 1 维数组输出。该 1 维数组包含按列最大顺序排列的 2 维数组元素。

特点

直通	有
采样时间	由驱动模块继承
标量扩展	不适用
向量化	有
零点穿越	无

6.7.7　Polynomial 模块

Polynomial 模块的属性对话框如图 6.92 所示。

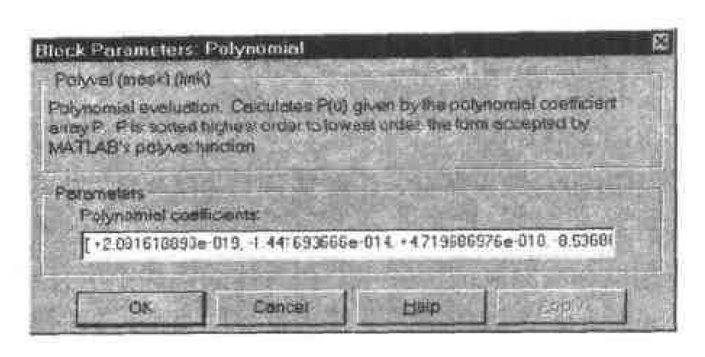


图 6.92 Polynomial 模块的属性对话框

说明：该模块通过定义系数参数来计算输入的实数多项式值。

可以使用 MATLAB 中的 polyval 指令来设置多项式的系数。该模块在每一仿真步计算多项式的值。模块输入和多项式的系数都必须是实数。

数据类型 接受单精度和双精度的信号。Polynomial coefficients 参数的类型必须与输入信号相同，同时，输出的类型与输入一致。

参数 Polynomial coefficients：定义多项式的系数。第一个元素对应于 xN，其他按照多项式的降幂排列。

特点

直通	有
采样时间	由驱动模块继承
标量扩展	无
向量化	有
零点穿越	无

6.7.8 PreLook-Up Index Search 模块

PreLook-Up Index Search 模块的属性对话框如图 6.93 所示。

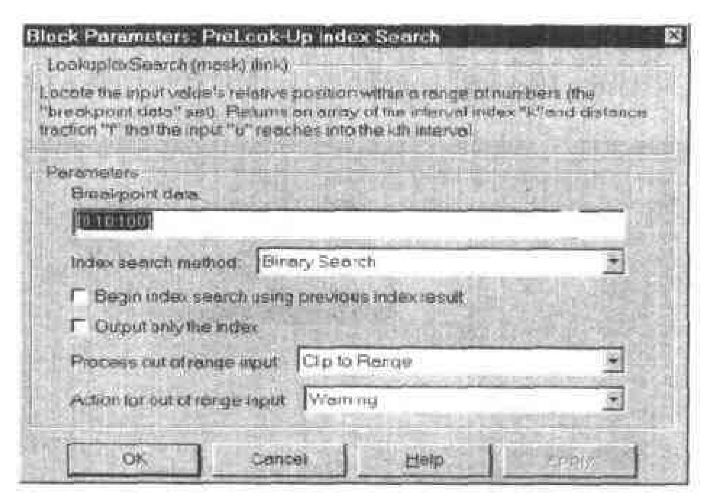


图 6.93 PreLook-Up Index Search 模块的属性对话框

说明：该模块能够计算输入在 Breakpoint data 参数值中的索引和相对偏移。

在使用该模块前必须定义一系列的断点值。模块输出包括两个元素，第一元素显示这些

断点值中不大于输入的最大值的索引，第二个元素显示输入在该索引和下一个索引之间的相对位置，用百分比表示。例如，如果断点值为：

[0 5 10 20 50 100]

如果输入为 55，则输出的信号（索引、相对偏移）为[4,0.1]，并在图标中用 k 和 f 显示。注意这里的索引是从零开始的。

数据类型　接受单精度和双精度的信号，但在具体的模块中所有输入必须是同类型的。Breakpoint data 参数值必须与输入信号同类型，同时输出的数据类型与输入一致。

参数　Breakpoint data：定义查询的断点集。

Index search method：指定索引搜索的方法。包括按位搜索（Binary search），等间隔搜索（evenly spaced points）和线性搜索（linear search）等。其中线性搜索对于输入变化不大的模块执行效率相对较高。

Begin index search using previous index result：指定该模块用前一时刻得到的索引值作为当前开始搜索的出发点，因此，对于变化不大的输入信号，可以提高索引搜索的效率。

Output only the index：如果该模块没有被用来驱动 Interpolation（n-D）Using PreLook-Up 模块，则输出的相对偏移量被忽略，而只输出 uint32 类型的索引值。

特点
直通	有
采样时间	由驱动模块继承
标量扩展	有
向量化	有
零点穿越	无

6.7.9　Interpolation（n-D）Using PreLook-Up 模块

Interpolation（n-D）Using PreLook-Up 模块的属性对话框如图 6.94 所示。

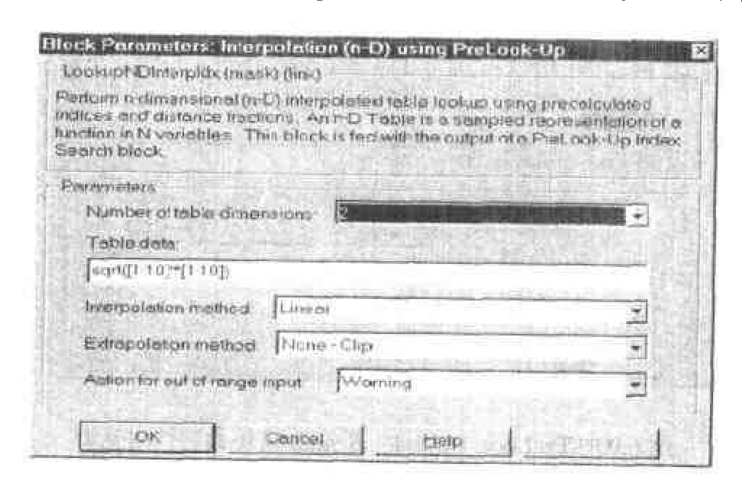


图 6.94　Interpolation（n-D）Using PreLook-Up 模块的属性对话框

说明：该模块利用 Look-Up Table（n-D）模块计算得到的索引和偏移量来完成 Look-Up

Table （n-D）模块的相同功能。使用该模块可以用一系列的 PreLook-Up Index Search 模块集来驱动多个 Interpolation 模块。对于需要大量插值的模型，使用这种方法能够提高仿真的质量。

该模块支持两种插值方式：常数间隔查询和线性插值。它们可以用于1维，二维或更高维的查询表。

可以在Table data参数域中指定输出值。这些查询表数据必须与PreLook-Up Index Search模块中的断点值一致。该模块通过PreLook-Up Index Search模块的输出在查询表中进行搜索和插值，从而得到模块的输出。

该模块的输出计算分为两类：

如果输入恰好满足断点参数值，则输出是查询表中相应行、列和更高维所对应的数据。

如果输入不满足任何断点参数值，则模块通过插值或外推来得到相应的输出。

数据类型　接受单精度和双精度的信号，但在具体的模块中所有输入必须是同类型的。Table data 参数值必须与输入信号同类型，同时输出的数据类型与输入一致。

参数　Number of table dimensions：指定查询表的维数。它指定了表中独立下标的个数，从而也间接指定了输入端口的数目。

Table data：定义查询表的内容。

Interpolation method：指定插值的方式。

Extrapolation method：指定外推计算的方式。

Action for out of range input：指定索引越界的动作，包括警告或错误等等。

特点　直通　有

采样时间　由驱动模块继承

标量扩展　有

零点穿越　无

6.7.10　S-Function 模块

S-Function 模块的属性对话框如图 6.95 所示。

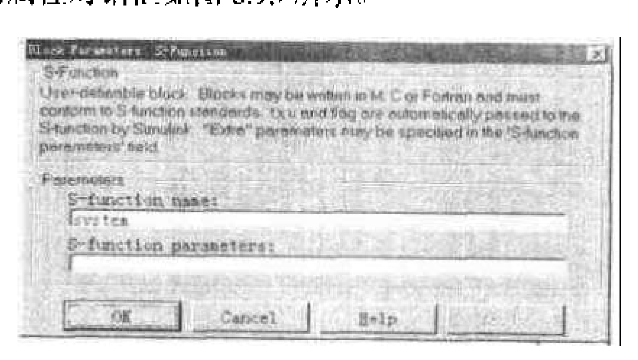


图 6.95　S-Function 模块的属性对话框

说明：该模块提供从方框图中访问 S 函数的方法，S-Function name 参数可以是 S 函数的 M 文件名或 MEX 文件名。

该模块允许直接传递附加参数给 S 函数。函数的参数可以指定为 MATLAB 的表达式或用逗号分开的变量。例如：

A, B, C, D, [eye(2,2);zeros(2,2)]

尽管单个的参数可以用方括号括起来，但是参数列表不能用方括号括起来。

该模块的图标显示 S 函数的名称，不论它包含的子系统有多少个输入和输出，通常只显示一个输入和一个输出端口。

当 S 函数包含有多个输入或者多个输出时使用向量信号线。输入向量的宽度必须与包含在 S 函数中的输入的数目相等。模块将第一个输入传递给 S 函数的第一个输入，将第二个输入传递给 S 函数的第二个输入，以此类推。输出向量的宽度与 S 函数的输出数目相等。

数据类型　依赖于 S 函数模块的执行。

参数　S-function name：定义所使用的S函数的名称。

S-function parameters：定义 S 函数的附加参数。

特点　直通　依赖于 S 函数模块的具体内容

采样时间　依赖于 S 函数模块的具体内容

标量扩展　依赖于 S 函数模块的具体内容

向量化　依赖于 S 函数模块的具体内容

零点穿越　无

6.8　Signals & Systems 库

6.8.1　Bus Selector 模块

Bus Selector 模块的属性对话框如图 6.96 所示。

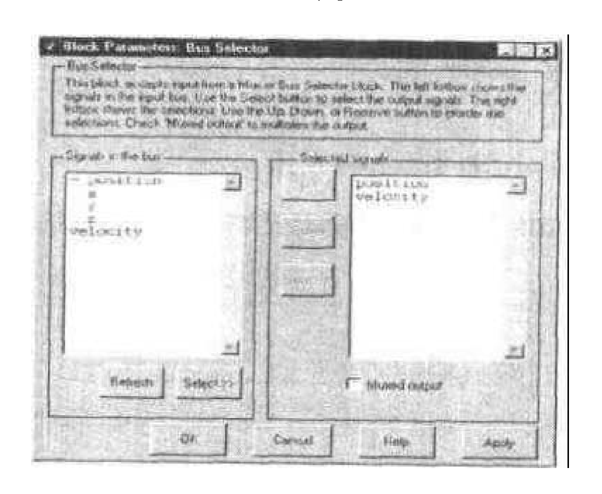


图 6.96　Bus Selector 模块的属性对话框

说明：该模块从一个 Mux 模块或另外一个 Bus Selector 模块接受信号。模块有一个输入端口，输出端口的数目取决于 Muxed output 选择框的状态。如果选中，则信号被组合于输出端口，并且只有一个输出端；否则，每个选择信号都对应于一个输出端口。

数据类型　接受和输出任何类型的实数或复数信号。

参数　Signals in bus：总线上的信号列表框中显示输入总线的信号。用“Select>>”

按钮选择从列表框中选择的输出信号。

Selected signals：选择的信号列表框中显示输出信号，可以用“Up”、“Down”和“Remove”按钮对信号进行排序。

6.8.2　Configurable Subsystem 模块

说明：*该模块可以表达任何包含于指定模块库中的模块，其对话框可以指定表达哪个模块，以及所表达模块的参数值。*

该模块简化了表达设计类型的创建。假设要创建一个汽车模型，则提供发动机选择。为这样的设计建模，首先创建一个可以用于不同汽车的发动机的类型库，然后在汽车模型中使用一个可配置子系统模块来表达发动机选择。要创建不同的基本汽车模型，只要使用可配置发动机模块对话框，就可以选择发动机。

该模块的外观随着代表的模块而变化。初始情况下，该模块不代表任何东西，此时，它没有端口。如果选择了一个库和模块，该模块显示图标和一系列与所选库中的输入和输出端口相应的输入和输出端口。SIMULINK 映射库端口与可配置子系统模块端口的原则为：

库中每一个惟一命名的输入或输出端口映射可配置子系统中独立的相同名字的输入或输出端口。

库中所有相同名字的输入或输出端口映射可配置子系统中同样的输入或输出端口。

用 Terminator 或 Ground 模块终止在当前选择的库模块中没有使用的任何输入输出端口。

该模块不提供与非输入和输出端口相应的端口，如触发和激化子系统中的触发和激活端口。因此，不能使用它来直接表达具有这些端口的模块。但可以间接地包装该模块于一个子系统中，而该子系统可以具有非输入或输出端口。

数据类型　接受和输出信号类型，与其表达的模块接受和输出的信号类型一致

参数　　　根据其当前是否表达了一个库及哪个模块而不同。初始没有表达任何东西时，该模块的属性对话框仅显示一个空的库名称（Library name）参数。

特点　　　具有当前代表的模块的特点，双击图标可显示当前代表模块的属性对话框。

6.8.3　Data Store Memory 模块

Data Store Memory 模块的属性对话框如图 6.97 所示。

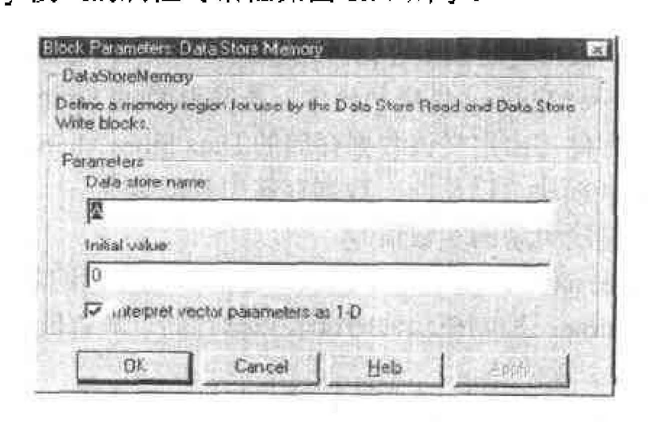


图 6.97　Data Store Memory 模块的属性对话框

说明：该模块定义并初始化一个名为共享数据存储器，它是一个对 Data Store Read 和 Data Store Write 模块有用的存储区域。

每一次数据存储都必须用该模块定义。它决定了 Data Store Read 和 Data Store Write 模块能否访问数据存储。

如果该模块在顶级模型中，数据存储可以被模型中任何地方的 Data Store Read 和 Data Store Write 模块访问；

如果该模块在子系统中，数据存储只能被模型中同一子系统或在模型层次中比该子系统更低的子系统中的 Data Store Read 和 Data Store Write 模块访问。

可以通过指定 Initial value 参数的值来初始化数据存储。值的长度决定了数据存储的长度。如果 ata Store Write 模块没有写入相同长度的数据就会出现错误。

数据类型　存储双精度类型的实数信号。

参数　Data store name：定义数据存储的名字。缺省值为 A。

Initial value：数据存储初始值。缺省值为 0。

Interpret vector parameters as 1-D：如果为选中状态，则 Initial value 参数的行或列矩阵初始化存储到 1 维数组(向量)中，其元素与行或列矩阵的元素相同。

特点　采样时间　不适用

向量化　可以

6.8.4　Data Store Read 模块

Data Store Read 模块的属性对话框如图 6.98 所示。

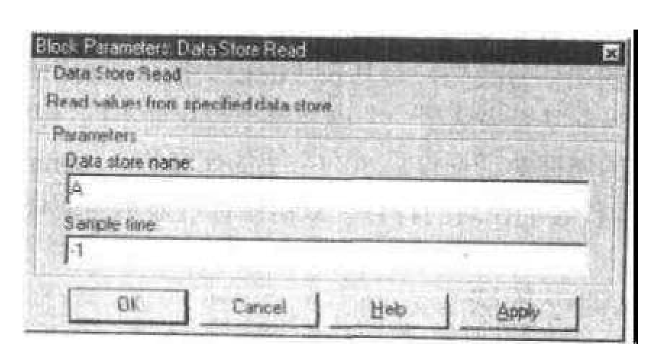


图 6.98　Data Store Read 模块的属性对话框

说明：该模块从一个被指定的数据存储中读取数据，并将数据传送给输出。数据存储中的数据，事先由 Data Store Memory 模块初始化，并且由 Data Store Write 模块写入数据。

要读取数据的数据存储，由定义该数据存储的 Data Store Memory 模块的位置确定。

多个 Data Store Read 模块可以从同一数据存储中读取数据。

数据类型　输出双精度类型的实数信号。

参数　Data store name：模块将要读取数据的数据存储的名字。

Sample time：控制模块何时读取数据存储。缺省值为-1，表示采样时间是继承的。

特点　采样时间　连续或离散

向量化　可以

6.8.5 Data Store Write 模块

Data Store Write 模块的属性对话框如图 6.99 所示。

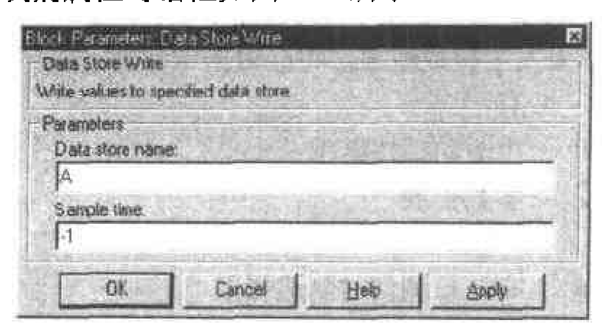


图 6.99 Data Store Write 模块的属性对话框

说明：该模块将其输入写入被指定的数据存储。每一次写入操作都会覆盖以前的内容。要写入数据的数据存储由定义该数据存储的 Data Store Memory 模块的位置确定。数据存储的大小由定义并初始化该数据存储的 Data Store Memory 模块设定。每一个写入数据给数据存储的 Data Store Write 模块必须写入相同数量的数据。

多个 Data Store Write 模块可以向同一个数据存储中写入数据。然而，如果在同一仿真步有两个 Data Store Write 模块试图给同一个数据存储写数据，结果将不可预测。

数据类型　接受双精度类型的实数信号。

参数　Data store name：模块将要写入数据的数据存储的名字。

Sample time：控制模块何时写数据存储，缺省值为-1，表示采样时间是继承的。

特点　采样时间　连续或离散

向量化　可以

6.8.6 Data Type Conversion 模块

Data Type Conversion 模块的属性对话框如图 6.100 所示

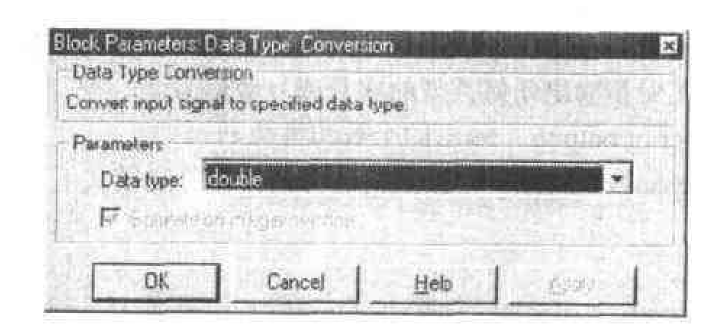


图 6.100 Data Type Conversion 模块的属性对话框

说明：该模块将其输入信号转换为由 Data type 参数指定的数据类型。

数据类型　输入可以是任何类型的实数或复数信号，如果是实数，输出也是实数；如果

输入是复数，输出也是复数。

参数　Data type：指定输入信号转换成何种类型的数据。选择 auto，将其输入信号转换成与其输出端口相连模块输入端口所需要的类型。

Saturate on integer overflow：只有在整数输出时，该参数才有效。如果选中该参数，如果发生整数溢出，将会输出饱和值。

特点　直通　有

采样时间　由驱动模块继承

标量扩展　参数扩展

向量化　可以

零点穿越　有，检测何时到达极限

6.8.7 Demux 模块

Demux 模块的属性对话框如图 6.101 所示。

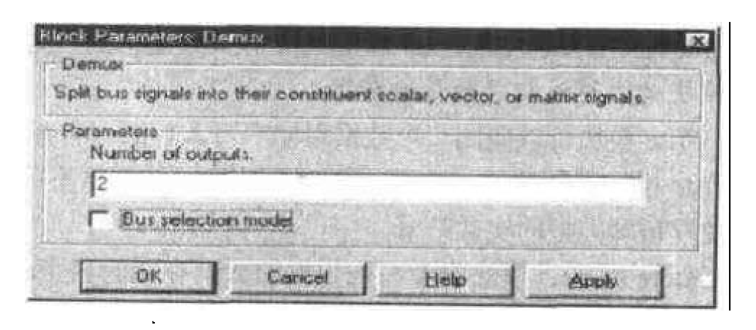


图 6.101　Demux 模块的属性对话框

说明：该模块将输入向量信号分解为若干输出连线，输出连线可以传输标量或者向量信号。SIMULINK 通过输出个数（Number of outputs）参数确定输出信号的数目和各个输出信号的宽度。

该模块既可以工作在向量模式也可以工作在总线模式下，这通过 Bus selection mode 参数进行设置。在两种工作模式下，模块接受的信号是不同的：向量模式仅仅接受标量、向量或行或列向量（只有一行或一列的二维数组），而总线模式只接受 Mux 或另一个 Demux 模块的输出信号。参数 Number of outputs 可以决定不同工作模式下的输出数目和维数。

数据类型　可以接受和输出任何类型的实数或复数信号。

参数　Number of outputs：输出的个数和维数。

Bus selection mode：设置模块是否工作在总线模式下。

6.8.8 Enable 模块

Enable 模块的属性对话框如图 6.102 所示。

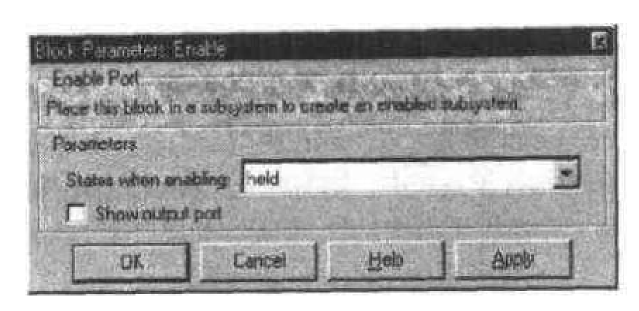


图 6.102 Enable 模块的属性对话框

说明：给子系统增加 Enable 模块使之成为使能子系统，该子系统只有在其使能输入端口的信号大于 0 时才运行。

在仿真开始时，SIMULINK 初始化使能子系统内的状态为它们的初始状态。当使能子系统重新启动时，参数 States when enable 可以确定包含在使能子系统内的各个模块的状态。

reset：将状态复位到它们的初始状态（缺省为 0）。

held：保持状态为它们以前的值。

可以通过选择 Show output port 选择框来输出使能信号。

一个子系统最多只能包含一个 Enable 模块。

数据类型　输入可以是任何的数据类型。

参数　States when enable：指定当子系统重新启动时，内部状态的处理方式。

　　　Show output port：如果选中该项，则该模块显示一个输出端口。

特点　采样时间　由使能信号确定

　　　向量化　可以

6.8.9 From 模块

From 模块的属性对话框如图 6.103 所示。

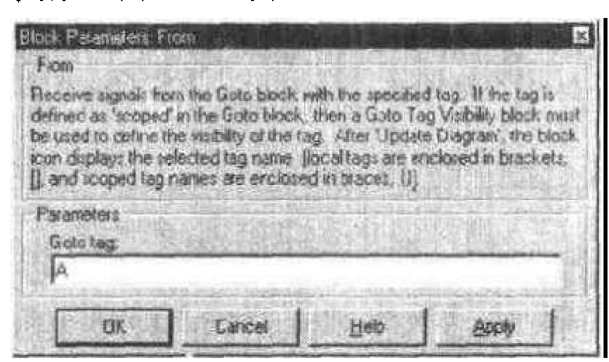


图 6.103 From 模块的属性对话框

说明：该模块从相应的 Goto 模块接受信号，然后将它传出作为它的输出。使用 From 和 Goto 模块可以从一个模块传送信号给另外一个模块而不用使用信号线。每一个 From 模块都与一个 Goto 模块相对应。Goto 模块的输入传给 From 模块，From 模块又传给与它相连的模块。要将一个 Goto 模块与一个 From 模块相关联，在 Goto tag 参数域中输入 Goto 模块的标记符。

一个 From 模块只能从一个 Goto 模块接受信号，而一个 Goto 模块可以传送信号给多个 From 模块。

相关联的 Goto 模块和 From 模块可以出现在模型当中的任何地方，但以下情况例外：如果两个模块中的一个在条件执行子系统中，则另外一个模块必须在同一个子系统中或者在它的下一层子系统（不是另一个条件子系统）中。但是，如果一个 Goto 模块与一个状态端口相连，信号可以传送给另一个条件子系统中的 From 模块。

Goto 模块标记符的可见性决定了能够接受它的信号的 From 模块。模块的图标表明了 Goto 模块标记符的可见性。

局部标记符的名字在方括号中（[ ]）;

范围标记符的名字在大括号中（{ }）;

全局标记符的名字显示时没有另外的符号。

数据类型　输出任何类型的实数或复数信号。

参数　Goto tag：传送信号给 From 模块的 Goto 模块的标记。

特点　采样时间　由驱动 Goto 模块的模块继承

向量化　可以

6.8.10　Function-Call Generator 模块

Function-Call Generator 模块的属性对话框如图 6.104 所示。

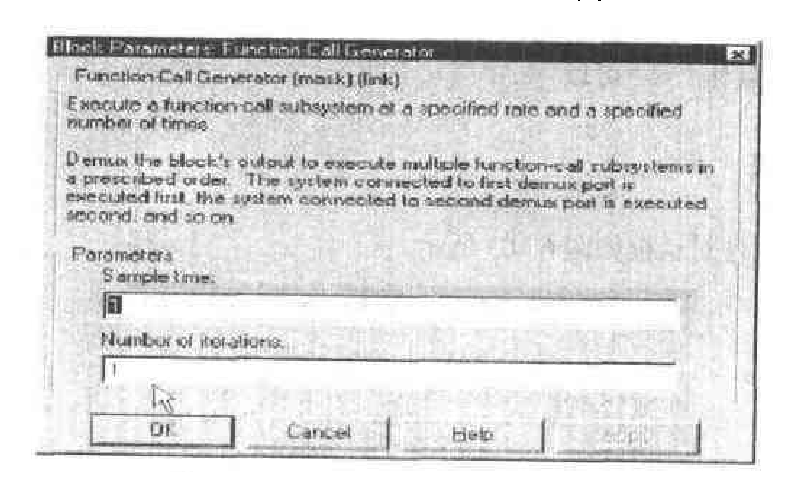


图 6.104　Function-Call Generator 模块的属性对话框

说明：该模块以 Sample time 参数指定的速率执行函数调用子系统。要以指定顺序执行多个函数调用子系统，首先将 Function–Call Generator 模块与 Demux 模块相连，Demux 模块的输出端口数与控制的函数调用子系统数一样。

数据类型　输出双精度类型的实数信号。

参数　Sample time：指定采样的速率。

Number of iterations：指定每一仿真步执行的迭代次数。

特点　直通　无

采样时间　由用户指定

标量扩展　无

6.8.11　Goto 模块

Goto 模块的属性对话框如图 6.105 所示。

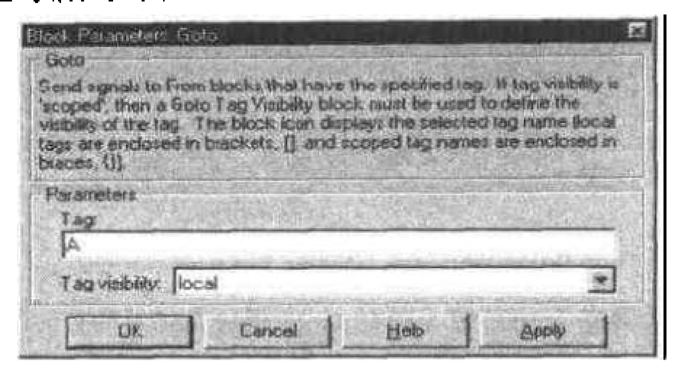


图 6.105　Goto 模块的属性对话框

说明：该模块传送它的输入给与它相关联的 From 模块。使用 From 和 Goto 模块可以将信号从一个模块传给另一个模块而不用信号线相连。

尽管一个 From 模块只能从一个 Goto 模块那里接受信号，但是一个 Goto 模块可以将它的输入信号传给多个 From 模块。Goto 模块将它的输入传给一个与它相关联的 From 模块就像它们在物理上连接在一起。Goto 模块和 From 模块通过使用 Goto 标记符相匹配。

Tag visibility 参数决定了获取信号的 From 模块的位置是否受到限制。

- local（缺省值），意味着使用标记符的 From 和 Goto 模块必须在同一个子系统中，标记符的名称用方括号（[ ]）括起来。
- scoped：意味着使用标记符的 From 和 Goto 模块必须在同一个子系统中，或者在模型层次中任何低于 Goto Tag Visibility 模块的子系统。范围标记符的名字用大括号（{ }）括起来。
- global：意味着使用标记符的 From 和 Goto 模块可以在模型的任何地方。

如果使用相同标记符名字的 From 和 Goto 模块在同一个子系统中时使用局部标记符。如果使用相同标记符名字的 Goto 和 From 模块在不同的子系统中时必须使用全局标记符。如果定义一个标记符为全局的，所有使用这一标记符的模块获取同一信号。一个定义为 scope 的标记符可以在模型中的多个地方使用。

数据类型	接受任何类型的实数或复数信号	
参数	Tag：指定 Goto 模块的标示符。	
	Tag visibility：指定 Goto 模块标记的范围：local、scoped 或 global。	
特点	采样时间	由驱动模块继承
	向量化	可以

6.8.12　Goto Tag Visibility 模块

Goto Tag Visibility 模块的属性对话框如图 6.106 所示。

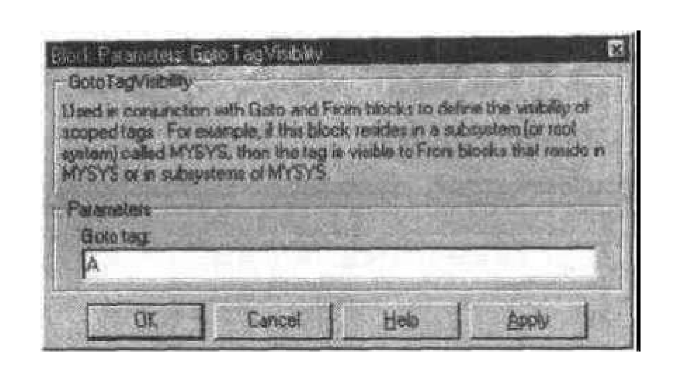


图 6.106　Goto Tag Visibility 模块的属性对话框

说明： 该模块定义具有 scoped 可见性的 Goto 模块标记符的可访问性。指定为 Goto tag 参数的标记符能够被与 Goto Tag Visibility 模块包含在同一个子系统中的 From 模块或者在模型层次中比包含有 Goto Tag Visibility 模块的子系统层次更低的子系统中的 From 模块。

对于参数 Tag visibility 的值是 scoped 的 Goto 模块来说，Goto Tag Visibility 模块是必须的。当标记符可见性是 local 或 global 时不需要该模块。该模块图标显示了用大括号括起来的标记符的名字。

参数　　　Goto Tag：Goto 模块的标记。

6.8.13　Ground 模块

说明： 可以用来连接那些输入端口没有与其他的模块相连的模块。如果有模块的输入端口悬空时运行仿真，SIMULINK 将会出现警告信息。使用 Ground 模块将那些模块接地以避免警告信息。Ground 模块输出的信号值为 0。

数据类型　输出与其相连端口的数据和数值类型一致的信号。

特点　　　采样时间　由驱动模块继承

　　　　　向量化　　可以

6.8.14　Hit Crossing 模块

Hit Crossing 模块的属性对话框如图 6.107 所示

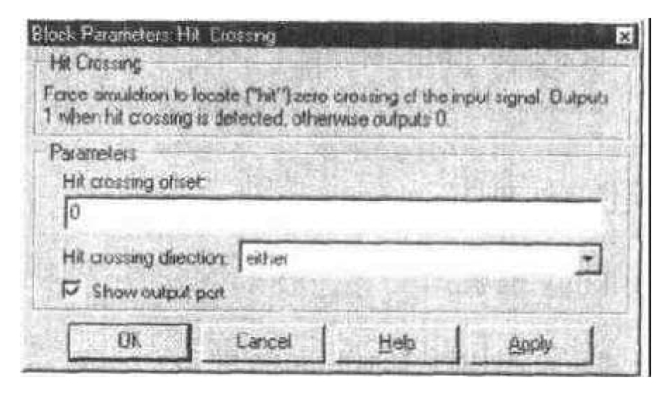


图 6.107　Hit Crossing 模块的属性对话框

说明：该模块检测输入是何时按照 Hit crossing direction 参数给出的方向到达了由 Hit crossing offset 参数给定的值。该模块寻找在机器误差容许范围内的穿越点。

该模块有一个输入。如果选取了 Show output port 选择框，模块的输出表明穿越是什么时候发生的。如果输入信号在某一时间步的值刚好等于偏移值，模块在那一时间步的输出值为1。如果输入信号的某相邻两点将偏移值包括在中间，模块在第二个时间步输出1。如果没有选取 Show output port 选择框，模块确保仿真找到穿越点但并不产生输出。

Hit Crossing 模块的作用就像一个“几乎等于”模块，在数学和计算机精度有限时计算圆整容限比较有用，该模块比在模块中加入逻辑以检测这一状态更为方便。

在 hardstop 和 cluch 演示例子中说明了如何使用 Hit Crossing 模块。在 hardstop 示例中，Hit Crossing 模块包含在 Friction Model 子系统中。在 clutch 示例中，Hit Crossing 模块在 Lookup Detection 子系统。

数据类型　输出逻辑类型信号，在逻辑兼容模式下，输出双精度信号。

参数　Hit crossing offset：检测穿越的值。

Hit crossing direction：输入信号接近穿越值的方向。

Show output port：如果选中，绘制一个输出端口。

特点　直通　有

采样时间　由驱动模块继承

标量扩展　可以

向量化　可以

零点穿越　有，检测穿越

6.8.15 IC 模块

说明：该模块设置与它的输出端口相连的信号的初始状态。

IC 模块对循环中的代数状态变量提供初始估计

数据类型　接受和输出双精度类型的信号

参数　Initial value：信号的初值。

特点　直通　有

采样时间　由驱动模块继承

标量扩展　参数扩展

状态个数　0

向量化　可以

零点穿越　无

6.8.16 Inport 模块

Inport 模块的属性对话框如图 6.108 所示。

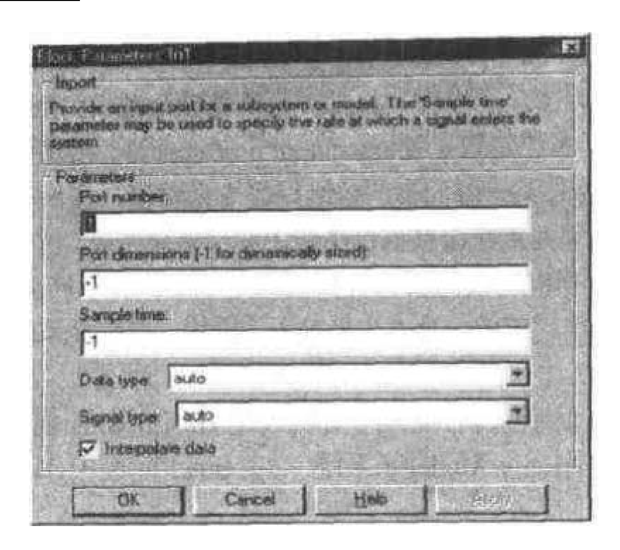


图 6.108　Inport 模块的属性对话框

说明：该模块从一个系统的外部引入信号到系统的内部。

SIMULINK 根据下面的规则指定 Inport 模块端口的序号。

在顶级系统或者子系统中从 1 开始自动地依次给 Inport 模块编号。

如果增加一个 Inport 模块，SIMULINK 将下一个可用的号码赋给它。

如果删除一个 Inport 模块，另外的端口将会自动地重新编号以确保 Inport 模块按顺序编号并且中间不会有遗漏的号码。

如果拷贝一个 Inport 模块到系统，它的端口号码不会被重新编号，除非它的号码与系统中已有的 Inport 模块相冲突。如果拷贝的 Inport 模块的端口号码不是按顺序的，必须重新给模块编号否则在运行仿真或更新系统时会得到出错的消息。

如果 Inport 模块提供一个向量信号，可以通过 Port width 参数指定 Inport 模块的输入宽度或者将 Port width 设为-1（缺省值），让 SIMULINK 自动地确定输入宽度。

Sample time 参数是信号进入系统的速率。缺省值为-1，使得模块从驱动它的模块那里继承采样时间。在一个顶级系统或在某些模型中，它的驱动 Inport 模块的采样时间没有确定时，最好为 Inport 模块设置 Sample time 参数。

（1）子系统中的 Inport 模块。子系统中的 Inport 模块代表子系统中的输入。信号到达 Subsystem 模块的一个端口时从那个子系统中与之相关的 Inport 模块流出。与 Subsystem 模块的输入端口相关联的 Inport 模块的 Port number 参数与该 Sunsystem 模块的输入端口的相对位置相一致。例如，Port number 参数为 1 的 Inport 模块从与它所在的 Subsystem 模块的最上面的一个端口相连的模块那里得到信号。

如果给 Inport 模块的 Port number 重新编号，该模块将与另外一个输入端口相连，尽管该模块继续接受子系统外的同一个模块的信号。

Inport 模块的名字显示在 Subsystem 模块的图标中作为端口的标签。要禁止显示标签，选取 Inport 模块并从“Format”菜单中选择“Hide Name”项，然后从“Edit”菜单中选择“Update Diagram” 菜单项。

（2）在顶级系统中的 Inport 模块。在顶级系统中的 Inport 模块有两个用处：从工作空间中提供外部输入，可以使用 Simulation Parameters 对话框或者 sim 指令实现；提供扰动模型的方法。

要提供从工作空间的外部输入，使用 Simulation Parameters 对话框，或者 Sim 指令的 ut 参数。

如果指定时间和输入值的矩阵，第一列应该是时间值得向量，剩下的列是在那些时间点处的数据，每一列按照对应的端口号码为一个 Inport 模块提供数据。如果有必要，在时间向量中没有的时间点处，SIMULINK 使用插值得到输入数据。

要通过 linmod 和 trim 分析函数提供一个对模型进行扰动的方法。Inport 模块定义在那些地方将输出引入模块。

数据类型 接受任何类型的实数或复数信号。

参数 Port number：输入模块的端口数。

Port width：输入端口输入信号的宽度。指定为-1，自动选择。

Sample time：信号进入系统的速率。

Data type：外部输入的数据类型。仅用于顶级输入端口。在子系统的输入端口对话框中不会出现。

Signal type：外部输入信号的实数或复数类型，仅用于顶层输入端口，在子系统的输入端口中不会出现。

Interpolate data：选择插值或外推的方法，仅用于顶层输入端口，在子系统的输入端口中不会出现。

特点 采样时间 由驱动模块继承

向量化 可以

6.8.17 Merge 模块

Merge 模块的属性对话框如图 6.109 所示。

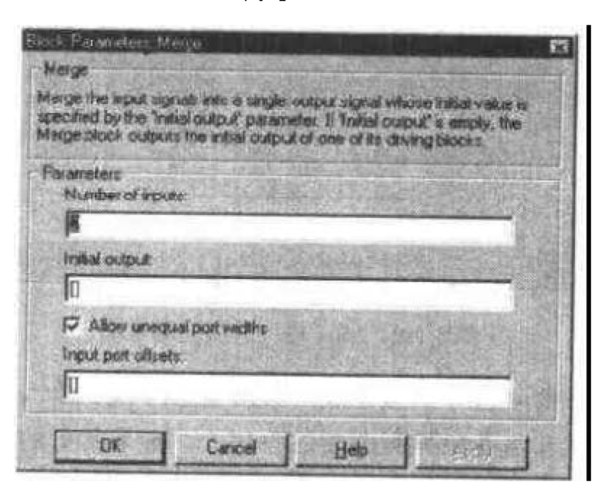


图 6.109 Merge 模块的属性对话框

说明：该模块将其输入连线合并为单个的输出线，其任何时刻的输出值与其驱动模块的最近计算输出相等。可以通过指定输入个数（Number of Inputs）参数，来指定该模块输入的个数。

如果该模块的 Allow unequal port widths 参数没有选中，则模块仅仅接受相等维数的信号，并且输出同样维数的数据。如果该项被选中，则该模块能够接受标量和具有不同元素个数的

向量（非矩阵），而且，可以指定每个输入信号相对输出信号开始处的偏移。输出信号的宽度为 max(w 1 +o 1 , w 2 +o 2 , ... w n +o n)，这里，w 1 , ... w n 是输入信号的宽度，o 1 , ... o n 是相应的偏移量。

可以通过 Initial Output 参数来指定模块的初始输出，如果该参数没有指定，同时存在多个驱动模块，则该模块的初始输出是驱动模块中最近计算的输出值。

数据类型　接受任何实数或复数和数据类型的信号，包括用户自定义的类型，如果输入类型为自定义类型，则初始条件必须为 0。

参数　Number of inputs：要合并的输入端口的个数。端口可以是标量或向量。

Initial output：模块的初始输出值，如果没有指定，与驱动模块的初始输出相等。

Allow unequal port widths：允许模块接受具有不同元素数目的输入。

Input port offsets：以向量形式指定各个输入相对于输出开始点的偏移。

特点　采样时间　由驱动模块继承

向量化　可以

6.8.18　Model Info 模块

说明：该模块将在模型的方框图中显示修改控制信息，作为模块的注释。

其属性对话框允许指定显示的内容和格式。

6.8.19　Mux 模块

Mux 模块的属性对话框如图 6.110 所示。

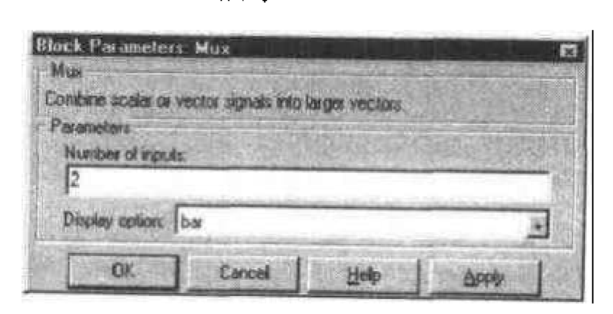


图 6.110　Mux 模块的属性对话框

说明：该模块将多个输入信号线组合成一条向量信号线。输入信号可以是标量、向量或矩阵。输出可以是向量也可以是矩阵和向量混合信号。如果所有的输入都是向量或类向量，输出也为向量。这里的类向量包括标量、向量和矩阵的行或列向量。如果任何一个输入不是类向量，则该模块输出总线信号。总线信号仅仅能够用来驱动虚拟模块，如：Demux、Subsystem 或Goto模块。

模块的Number of Inputs参数可以用来指定输入的名称、数目和维数。不同的输入格式具有不同的含义：

（1）标量

指定模块的输入数目，这时模块可以接受任何维数的信号，而且，SIMULINK为每个输

入命名signalN，这里的N表示输入端口号。

（2）向量

向量的长度规定了输入的个数，其中每一个元素对应于相应输入的维数，正数表示该输入只能输入指定维数的信号，而负数表示可以接受任何维数的向量或矩阵。

（3）元胞数组

元胞数组的长度规定了输入的个数，其中每一个元胞对应于相应输入的维数，标量N表示该输入只能输入N维的信号，而负数表示可以接受任何维数的信号。

（4）信号名列表

可以输入信号名称的列表。SIMULINK将每个名称对应到相应的端口和信号。例如如果输入“position,velocity”，则Mux模块有两个输入，并分别命名为position和velocity。

数据类型　接受任何类型的实数或复数信号，包括混合类型向量。

参数　Number of inputs：输入的个数和宽度。

Display option：在模型中显示模块的图标。

none：Mux出现在图标中。

names：在每个端口附近显示信号名。

bar：以实心前景颜色显示图标，缺省值。

6.8.20 Output 模块

Output 模块的属性对话框如图 6.111 所示。

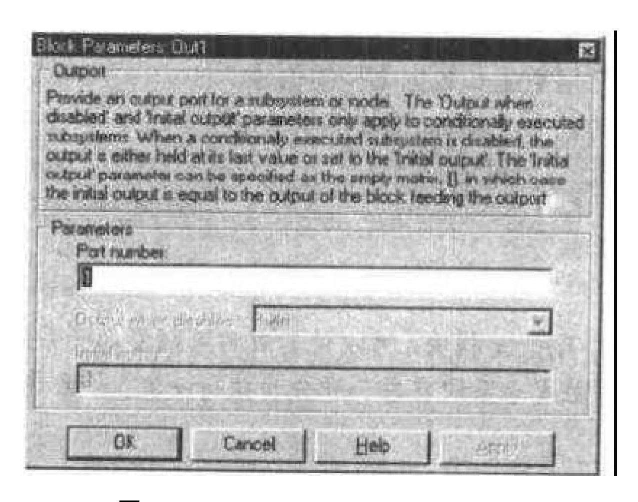


图 6.111 Output 模块的属性对话框

说明：该模块与Input模块相对应，其功能是完成从系统到系统外的目标的连接。详细内容可以参考Input模块的说明。

数据类型　接受任何数据类型的实数或复数信号。

参数　Port number：输出端口的数目。缺省值为1。

Output when disabled：对于条件执行子系统，当系统禁止时的输出。

Initial output：对于条件执行子系统，子系统禁止后执行前模块的输出。

特点　采样时间　由驱动模块继承

向量化　可以

6.8.21　Selector 模块

说明： 该模块将选取的输入向量或矩阵的元素作为它的输出。它既可以接受向量也可以接受矩阵信号。通过Input Type参数的设置，可以使Selector工作在向量选择模式或矩阵选择模式下，不同的工作模式其属性对话框也不同。

特点　采样时间　由驱动模块继承
　　　向量化　　可以

6.8.22　Subsystem 模块

说明： 该模块表示包含在另外一个系统中的子系统。可以通过三种方法来创建子系统。同时，子系统模块也是创建条件执行子系统的前提条件，具体用法可以参考本书第七章的内容。

特点　采样时间　依赖于子系统中的模块
　　　向量化　　依赖于子系统中的模块
　　　零点穿越　如果有使能或触发端口，则存在零点穿越。

6.8.23　Terminator 模块

说明： 该模块可以用来盖住那些输出端口没有与另外的模块相连的模块。如果对于包含有来被连接的输出端口的模型运行仿真时，SIMULINK 会发出警告消息，使用 Terminator 模块可以避免警告消息的出现。

数据类型　接受任何数值和数据类型的信号。
特点　　　采样时间　由驱动模块继承
　　　　　向量化　　可以

6.8.24　Trigger 模块

说明： 给子系统加入 Trigger 模块，可以使其成为可触发的子系统。当传过触发端口的信号以给定的方式改变时，触发子系统在每一积分步执行一次。子系统包含的 Trigger 模块不能多于 1 个。

可以改变 Trigger type 参数以选择触发子系统执行的事件的类型：

- rising：当控制信号从 0 升到一个正数值时触发子系统执行。
- falling：当控制信号从 0 下降到一个负数时触发子系统执行。
- eighter：当控制信号从 0 升到一个正数值，或从 0 下降到一个负数时触发子系统执行。
- function-call：使得子系统的执行受控于 S 函数的内部逻辑。

可以通过选取 Show output port 选择框输出触发信号。选取这一选项允许系统确定是哪类信号触发了触发器。信号的宽度是触发信号的宽度。信号值为：

- 1：对于导致上升沿触发的信号。
- -1：对于导致下降沿触发的信号。
- 0：其他情况。

数据类型　接受双精度或逻辑类型的信号。

参数　Trigger type：子系统触发执行事件的类型。

Show output port：如果选择该项，将在触发模块上绘制一输出端口，输出触发信号。

Output data type：指定触发输出的数据类型（双精度或8位整型）。

特点　采样时间　由触发端口信号确定

向量化　可以

6.8.25 Width 模块

说明：该模块输出输入信号的宽度。

数据类型　接受任何数据类型的实数或复数值信号，输出双精度类型的实数信号。

特点　采样时间　离散

向量化　可以

6.8.26 Probe 模块

Probe 模块的属性对话框如图 6.112 所示。

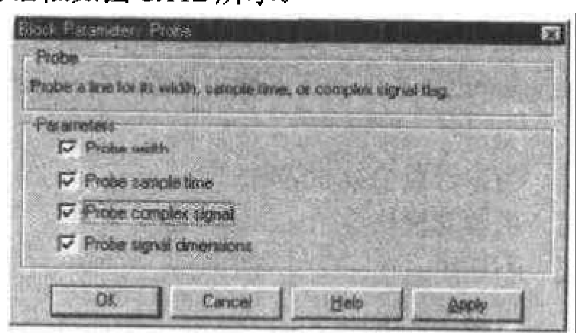


图 6.112　Probe 模块的属性对话框

说明：该模块选择性地输出其输入信号的信息。输出包括：输入信号的宽度、维数、采样时间和输入复数值标志。该模块只有一个输入端口，其输出端口数取决于所选择的探测信息。每一个探测值就是在不同端口上的一个独立的输出信号。仿真时该模块显示探测数据。

数据类型　接受和输出双精度型信号。

参数　Probe width：输出探测连线地宽度。

Probe sample time：输出探测连线地采样时间。

Probe complex signal：探测信号是复数，则输出 1，否则输出 0。

Probe signal dimensions：输出探测信号的维数。

特点　直通　当零极点个数相等时

采样时间　由驱动模块继承

标量扩展　可以

向量化　可以

零点穿越　无

6.8.27　Reshape 模块

Reshape 模块的属性对话框如图 6.113 所示。

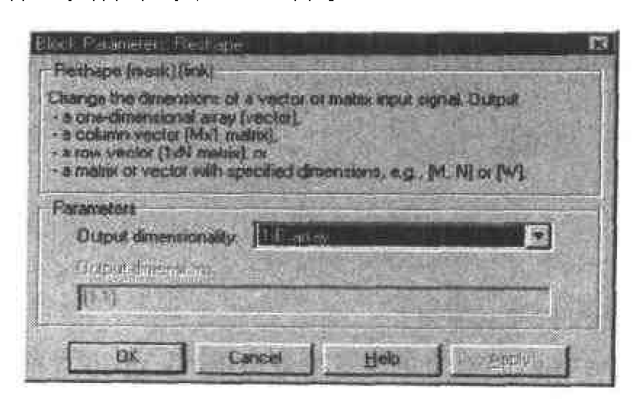


图 6.113　Reshape 模块的属性对话框

说明：该模块可以将输入信号的维数转换成 Output dimensionality 参数指定的维数。例如，可以使用该模块将 N 个元素的向量转换成 1 × N 或 N × 1 的矩阵。

数据类型　接受和输出任何类型的数据。

参数　　Output dimensionality：输出信号的转换方式，包括1-D array（1维数组）、Column vector（列向量）、Row vector（行向量）和Customize（定制）。

Output dimensions：当选择定制输出维数（Customize）时，指定具体的输出维数。

特点

直通	当零极点个数相等时
采样时间	由驱动模块继承
标量扩展	不适用
向量化	可以
零点穿越	无

6.8.28　Matrix Concatenation 模块

Matrix Concatenation 模块的属性对话框如图 6.114 所示。

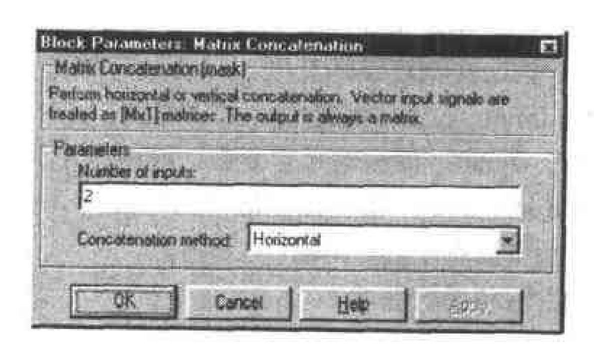


图 6.114　Matrix Concatenation 模块的属性对话框

说明：该模块沿行或列将输入矩阵 u1，u2，...，un 连接起来，其中 n 由 Number of inputs

参数指定。如果所有的输入都是基于采样的，则输出也是基于采样（sample-based）的，否则，输出是基于框架（frame-based）的。

（1）水平矩阵连接

当 Concatenation method 参数设置为 Horizontal 时，模块将沿行连接输入矩阵，即

$$y = [u1\ u2\ u3\ \ldots\ un]$$

对于水平矩阵连接，所有的矩阵都必须有相同的行数。输出矩阵的维数为 $M \times \sum N_i$，其中，N_i 为输入矩阵 $u\,i\,(i = 1, 2, \ldots, n)$的列数。

如果某些输入是 M 个元素的一维向量，而其他的输入是 M×N *l* 矩阵，则向量输入作为 M×1 矩阵对待。

（2）垂直矩阵连接

当 Concatenation method 参数设置为 Vertical,时，模块将沿列连接输入矩阵，即

$$y = [u1;\ u2;\ u3;\ \ldots;\ un]$$

对于垂直矩阵连接，所有的矩阵都必须有相同的列数。输出矩阵的维数为 $\sum M_i \times N$，其中，M_i 为输入矩阵 $u\,i\,(i = 1, 2, \ldots, n)$的列数。

如果某些输入是 M 个元素的一维向量，而其他的输入是 $M_i \times 1$ 矩阵，则向量输入作为 $M_i \times 1$ 矩阵对待。

（3）一维向量连接

如果该模块的所有输入都是一维向量，其元素个数分别为 N_i，则输出按照输入端口的顺序连接这些向量，最后输出 $\sum M_i \times 1$ 的矩阵。

参数　Number of inputs：指定输入的个数。

Concatenation method：选择输入连接的方式。

第 7 章　子系统的创建与封装

SIMULINK 在创建系统模型的过程中常常采用“分层”的设计思想。用户在进行动态系统的建模过程中，可以根据需要将模型中部分比较复杂，或者共同完成某一功能的基本模块（也可能是低一层次的子系统）封装起来，并采用一个简单的图形来替代表示。这样可使整个模型结构清晰、显示简单，让用户将主要精力放在系统级上的信号分析上。这种设计方法实际上也是面向对象设计思想的一种体现。

依照封装后系统的不同特点，SIMULINK 具有一般子系统、封装子系统和条件子系统等三种不同类型的子系统，下面我们将分别对各种子系统的基本特点和创建过程依次给予介绍。

7.1　子系统介绍

7.1.1　分层的建模思想

用户在进行动态系统的仿真过程中，常常会遇到比较复杂的系统。无论多么复杂的系统都是由众多不同的基本模块组成的。在采用 SIMULINK 创建系统模型时，当然希望将系统所有的模块都在同一个模型窗口中显示出来，然而对于比较庞大的系统，这时不现实的。另外，较大的系统一般包含大量的基本模块，用户一般并不关心这些基本模块之间的信号交互，一般只希望了解系统不同组成部分之间的信号流向。因此，有必要根据系统的结构，将同属于一个部分的基本模块封装起来，在模型窗口中用封装后的模块（一般很简单）来替代原来的部分。这种思想被称为“分层建模”的设计方法，在实际工作中经常用到。

SIMULINK 本身就体现了“分层建模”的思想。例如各种基本模块库可看成是封装了相关基本模块的子系统。采用这种设计思想的好处在于：

- 体现了面向对象的设计方法。封装后的子系统对用户而言是透明的，用户可将它看作一个“黑匣子”，只需要关心子系统两端的输入输出，而无论子系统内部信号的处理过程。
- 提高了工作效率和可靠性，封装后的子系统可实现“重用性”。
- 对于封装的子系统，其工作空间与基本工作空间相互独立，简化了模型的设计。
- 符合实际系统分层或分部组成的实际情况。

依照封装后系统的不同特点，SIMULINK 具有一般子系统、封装子系统、使能子系统和触发子系统等四种类型的子系统，下面我们将分别依次给予介绍。

7.1.2　用户模块库的定制

通过子系统的封装，用户可以很容易实现 SIMULINK 模块库的定制。将用户自己创建的模块封装起来，并给与封装后的模块库以适当的外观和参数，就完成了用户自己的模块库的定制，在使用上，定制的模块库与标准的 SIMULINK 模块库并无区别，这样，用户就用自己的模块库对 SIMULINK 进行了扩展，SIMULINK 这种广泛的开放性无疑得到了越来越多的用户的支持和喜爱。

7.1.3　条件子系统

在 SIMULINK 模块库中，有两个特殊的模块：使能模块（Enable）和触发模块（Trigger）。如果把这种模块放在某个子系统中，则该子系统是否发生作用就取决于外界的条件是否满足，这种系统我们称之为条件执行子系统。这种子系统为我们创建更加复杂系统的仿真模型提供了方便。

条件子系统又可分为使能子系统（Enable Subsystem）、触发子系统（Trigger Subsystem）以及触发使能子系统（Enable and Trigger Subsystem），第四节将详细介绍它的创建方法。

7.1.4　一个简单的例子

下面以封装子系统为例，列举一个简单的例子。通过这个例子，我们可以领略分层建模的基本思想。

假设一个由 y=mx+b 描述的系统，它的 SIMULINK 模型如图 7.1 所示。

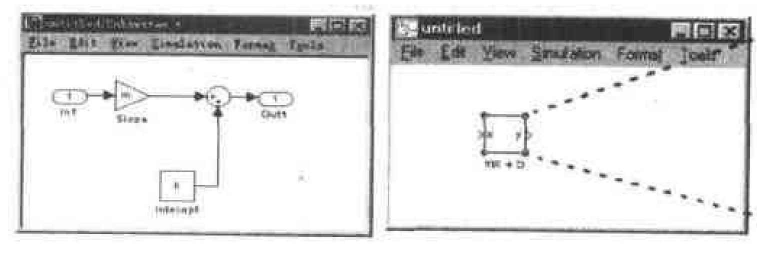


图 7.1　一个简单的子系统

当用户用鼠标双击子系统模块时，SIMULINK 一般会在一个新的模型窗口中显示该子系统封装前的模型结构。在这个例子中，封装的子系统包括一个名为 Slope、大小为 *m* 的增益模块和一个名为 Intercept、大小为 *b* 的定常模块。

在这个例子中，我们还为该子系统创建了一个定制的对话框和图标，如图 7.2 所示。

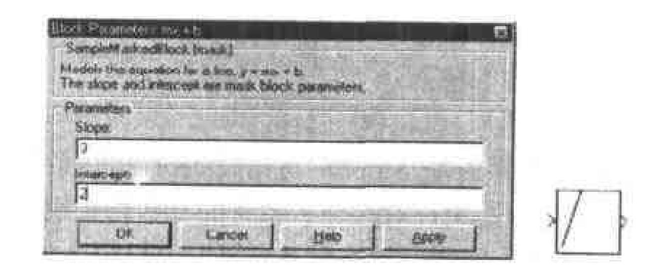


图 7.2　定制的封装子系统对话框和图标

在对话框中，我们可以分别输入 Slope 和 Intercept 模块的参数值，而图标则是封装后子系统的外观。

一旦我们在对话框中完成了相关参数的设置，SIMULINK 会自动将输入的数值映射到子系统中的某个具体参数上，对用户而言这个过程是不可见的。子系统原有的复杂性和多个参数的设置统统被封装起来，这样，大大简化了模型的复杂程度。而封装后的子系统与 SIMULINK 标准的模块库在外观和使用上并无区别。

下面来看看封装一个子系统需要做些什么工作：

- 定义封装子系统的提示（Prompts）。在这个例子中为 Slope 和 Intercept。
- 定义用来存储输入参数的变量名。
- 输入模块的功能描述和帮助文本。
- 输入用于图标绘制的指令。
- 输入可与图标绘制过程进行交互的命令（本例中没有）。

下面我们简要说明这些步骤是如何完成的。

1. 封装对话框的创建

为了实现对子系统的封装，我们可以选中一个子系统模块，然后单击菜单 “Edit: Mask Subsystem”。系统将弹出一个属性设置对话框，称之为封装编辑器（Mask Editor），如图 7.3 所示。

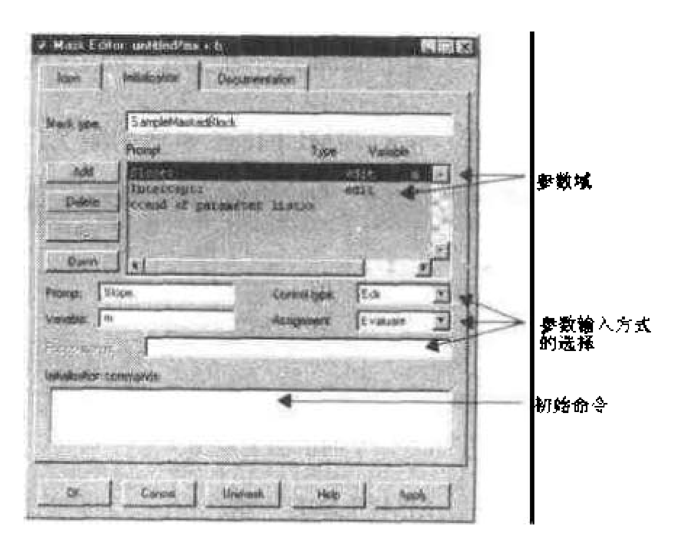


图 7.3　封装子系统的封装编辑器

在封装编辑器中，用户可以完成对封装子系统基本属性的设置。包括：

- 用来描述输入参数的文本标示；
- 用户以何种方式进行参数的输入；
- 储存输入参数的变量名。

在这个例子中与 slope 相联系的是 Slope，与 intercept 相联系的是 Intercept。slope 和 intercept 被设置成编辑框，这意味着用户对这个子系统的参数将采用编辑框的方式进行输入。输入的参数值被贮存在封装工作空间中，这个工作空间与 MATLAB 的基本工作空间是分离的。在这个例子中，对 slope 的输入将映射到变量 m，而对 intercept 的输入映射到变量 b，无

论是变量 m 还是变量 b 都只存在于封装子系统的封装工作空间中，这也是封装子系统的特点之一。如果子系统只是一般（Encapsulated）而没有实现封装，则这些变量存在于 MATLAB 的基本工作空间中。

2. 创建模块功能描述和帮助文本

模块功能描述是对封装子系统完成功能的一个简要说明，而帮助文本则详细说明了该子系统的一些信息。图 7.4 显示的是它们的设置画面。

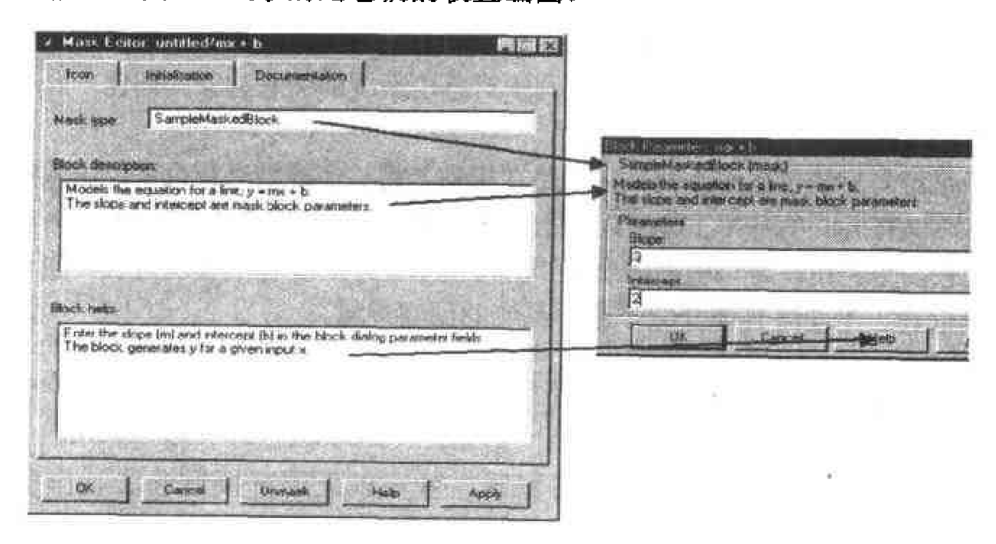


图 7.4　封装子系统模块功能描述和帮助文本

3. 创建模块图标

如果用户不对模块的外观进行设置，该模块就以标准的 SIMULINK 子系统图标进行显示。在这个例子当中，我们设置了新的图标。图标的设置是在相应的对话框中输入适当的绘图指令，如图 7.5 所示。在本例中，输入的指令为：

```
>>plot((0,1),(0,m)+(m<0))
```

该指令保证斜率为负时，绘制的指令仍在可视的范围内。这里需要注意的是，绘图指令所用到的变量也只局限于封装工作空间内部。

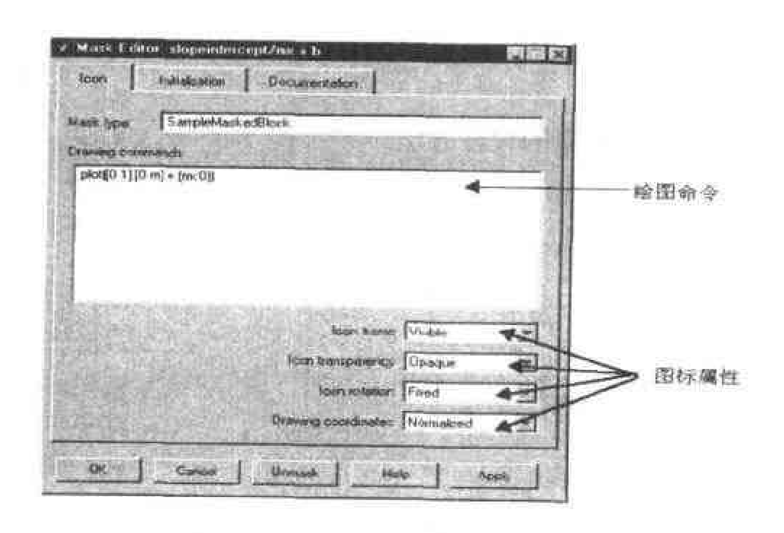


图 7.5　图标的定制

7.2 一般子系统

7.2.1 什么是一般子系统

根据前一节我们知道，在 SIMULINK 当中，如果被研究的系统比较复杂，那么直接用基本模块构成的模型就比较庞大，模型中的主要流向不容易辨认。此时，若把整个模型按照实现功能或对应物理器件划分成几块，将有利于对整个系统的概念抽象。创建一般子系统是其中最为简单的方法。

一般子系统与封装子系统不同，它只是在视觉上将整个模型进行分层，子系统内部的模块仍旧共享 MATLAB 的基本工作空间。由于不需要进行封装对话框的设置，一般子系统的创建过程比较简单。用户可以根据需要采用标准的 Subsystem 标准模块进行，也可以鼠标在模型上直接框选。下面将具体介绍一般子系统的创建步骤。

7.2.2 采用框选法创建一般子系统

如果用户已经建立起系统的模型方框图，就可以采用直接框选法制作一般子系统（Encapsulated Subsystem）。具体步骤为：

生成子系统：在模型窗口中，用鼠标将需要包含进子系统的模块框起来（在图中将以虚线框标示），单击菜单项“Edit: Create Subsystem”，便将框选的部分包装在一个名为 Subsystem 的模块中。包装后，新生成子系统中的模块和信号线可能会显得比较杂乱，我们可以通过鼠标操作进行整理。

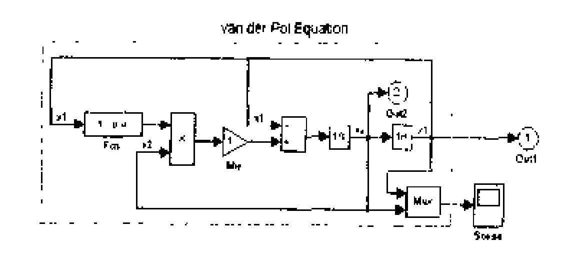
模块名的修改：用鼠标双击新生成子系统中的模块名，将模块名改为其他适当的名称。

输入输出端口的设置：双击新生成子系统的图标，引出该子系统的结构模型窗口。把该结构模型窗口中的输入口模块的缺省名改为其他适当的名称，把输出口模块的缺省名也改为其他适当的名称。

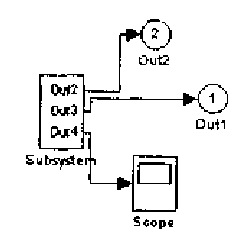
最后单击菜单项“File：Save”将以上创建的一般子系统保存下来。

以后用户打开该模型文件时，SIMULINK 将只显示一般的子系统，用户双击子系统图标，可以进一步观察其内部的构造。

直接框选法体现的是自下而上的设计方法。



(a) 用鼠标进行框选



(b) 创建后的子系统

图 7.6　用直接框选法创建一般子系统

7.2.3 采用 Subsystem 模块方法创建子系统

除了在模型中直接用鼠标框选的方法之外，还可以直接使用 SIMULINK 标准 Subsystem 模块的方法创建一般子系统。具体步骤如下：

建立总体模型：在新建的模型窗口中复制包括子系统模块在内的所有标准模块。

子系统的绘制：双击 Subsystem 模块，引出一个空白的模型窗口。在该窗口中完成子系统结构的绘制。然后单击菜单“File：Close”，关闭该窗口。于是原先的标准子系统出现相应的输入输出口，然后把相应的子系统名修改为适当的名称。这样就完成了一个一般子系统的绘制。

整个模型的绘制：完成各个子系统后，将模型窗口中的各种模块包括已经制作好的子系统用信号线连接起来。就形成了完整的仿真框图。

7.3 封装子系统

7.3.1 什么是封装子系统

一般子系统的操作过程很简单，并且在一定程度上提高了分析问题的抽象能力。然而，一般子系统仍然从 MATLAB 基本工作空间获取变量，因此没有实现完全意义上的封装。

在一般子系统的基础上对子系统进行封装，则可以解决这个问题。封装子系统在外表上与普通模块完全一样，有自己的参数对话框和图标，并且其中的变量有着与基本工作空间独立的工作空间，即封装工作空间。

封装子系统是在一般子系统的基础上进一步设置而成的，正如第一节所述，封装子系统还给用户提供了一种扩展 SIMULINK 模块库的方法。

7.3.2 封装子系统的创建过程

由于封装子系统是在一般子系统的基础上完成的，因此，首先我们必须创建一般子系统，下面具体论述封装子系统的创建步骤。

首先按照前一节方法构造一般子系统。

选中该一般子系统，单击模型窗口中的菜单项“Edit：Mask system”，打开封装编辑器。

在打开的封装对话框中设置封装模块的参数对话框：设置参数项、模块功能描述和帮助文本以及用户定义的模块图标。

如果要对已经存在的封装子系统进行修改，可以单击菜单项“Edit：Edit Mask”。

7.3.3 参数对话框的设置

封装子系统创建以后，需要为它定制一个用来进行相关参数进行设置的对话框，这个过程是在封装编辑器中完成的。下面我们介绍封装编辑器的 3 个属性页面的设置。

1. 初始化（Initialization）页面的设置。

封装界面可以让封装子系统的用户向封装子系统中的变量输入参数值。封装界面的创建是在初始化（Initialization）页面中设置的，如图 7.3 所示。

下面我们具体介绍各部分的含义。

（1）提示（Prompts）及相关变量

Prompts 是对用户所输入参数基本信息的简单描述。Prompts 在未来封装对话框中出现的次序与在封装编辑器中定义的次序相同。在定义一个 Prompts 之后，我们可以定义一个与之相关的用来存储相应参数的变量，包括参数的输入方式。Assignment 选项规定了该参数的接受方式，可以是变量形式（Evaluate），也可以采用字符串（Literal）。

对提示的操作包括：

- 添加一个新的 Prompts：单击“Add”按钮，在 Prompt 域中输入文本，在 Variable 域中输入变量名。
- 编辑一个 Prompts：用鼠标单击需要编辑的 Prompts，为变量名、输入控制类型等输入适当的值。
- 删除 Prompts：单击【Delete】按钮，被选中的 Prompts 将被删除。
- 移动 Prompts：选中需要移动的 Prompts，用【Up】和【Down】键可以调整 Prompts 的上下位置。

（2）控制类型（Control type）

SIMULINK 允许用户自由选择封装子系统中参数的输入方式，这些方式包括：编辑框、选择框以及弹出菜单等。图 7.7 显示的三种输入方式的情况。

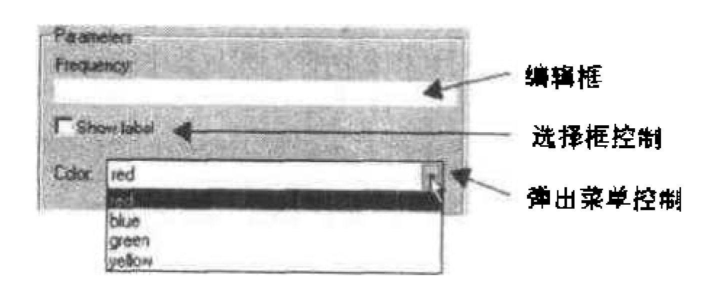


图 7.7　三种不同的控制类型

（3）可调参数（tunable parameter）

可调参数是在封装子系统中能够在仿真过程中实时调整的封装参数。封装子系统创建后，所有的参数都是可调的，用户可以通过 MaskTunableValues 参数的设置来允许或禁止相关参数的可调性。MaskTunableValues 是一组字符串数组，其中每个成员对应与封装子系统的输入参数。例如，第一个成员对应与第一个参数，第二个成员对应与第二个参数，以此类推。如果相应参数是可调的，则对应的 MaskTunableValues 值为 on，否则为 off。

下面的指令用来设置当前选择的封装模块的第一个参数为不可调状态。

```
ca = get_param(gcb, 'MaskTunableValues');
ca(1) = 'off'
set_param(gcb, 'MaskTunableValues', ca)
```

（4）初始化命令（Initialization Commands）

初始化命令是在封装模块初始化时执行的一系列指令，用户可以在其中完成封装模块内部一些变量的初始化工作。

SIMULINK 在以下情况下将执行初始化命令。

- 模块被载入。
- 仿真开始或相关的框图进行更新。
- 封装模块发生旋转。
- 模块图标被重画，而其中的绘图命令包含初始化命令中的参数。

注意，初始化命令只是在封装工作空间中有效，不能访问基本工作空间中的变量。

（5）封装工作空间

封装子系统在以下情况下将创建一个局部工作空间，称为封装工作空间。

- 封装过程包含初始化命令。
- 封装子系统定义了提示和与提示相关的变量。

封装模块不能访问MATLAB基本工作空间或其他封装工作空间。封装工作空间中的变量包括与封装参数相关联的变量和初始化命令中定义的变量。

封装工作空间只能被该封装模块访问。如果封装模块是一个子系统，则该封装工作空间能被其中所有的模块访问。封装工作空间与M函数创建的函数工作空间相类似，用户可以想想象封装模块是一个函数，封装编辑器输入的参数就如同是函数的输入参数，而该“函数”的局部工作空间就是封装工作空间。

（6）初始化命令的调试

可以采用下列方法进行初始化命令的调试：

- 在初始化命令的指令后面不加结束符号（分号），以使指令的计算结果能在命令窗口中显示出来。
- 在初始化命令中的适当位置放置keyboard指令，程序执行到该指令后将会把控制权交给用户，这时用户可以输入其他指令对执行情况进行分析。
- 在MATLAB命令窗口中输入下列指令：

```
>>dbstop if error
>>dbstop if warning
```

如果在初始化命令当中存在错误，程序就会自动停止执行，这时你可以检查封装工作子空间中的变量。

2. 图标属性页（Icon Page）的设置

图标属性页使用户可以定制封装模块的图标，如图 7.5 所示。图标的绘制是通过在画图命令域（Drawing commands field）内输入适当的指令实现的。创建的图标可以是描述文本、状态方程、图形以及图像等等。

画图指令可以访问封装工作空间中的所有变量，可以显示文本、一个或多个图形或者传递函数。如果在画图命令域输入多个指令，则绘制的最终结果以来于这些指令的先后顺序。

（1）在模块图标中显示文本

下列指令可以在模块图标区显示文本：

```
>>disp('text') 或disp(variablename)          %居中显示文本
```

>>text(x, y, 'text')　　%在(x,y)坐标显示文本

>>text(x, y, stringvariablename)

>>text(x, y, text, 'horizontalAlignment', halign, 'verticalAlignment', valign)　%在(x,y)坐标以垂直方式显示文本

>>fprintf('text') 或 fprintf('format', variablename)　　%以设定格式输出文本

>>port_label(port_type, port_number, label)

port_label命令用来指定图标中端口的标示。指令的用法是

>>port_label(port_type, port_number, label)

其中port_type为'input' 或 'output', port_number为一整数, label参数为指定标示的字符串。例如，下面的指令将输入端口1定义为a。

>>port_label('input', 1, 'a')

（2）在模块图标中显示图形

用户可以利用多个plot指令在封装模块的图标中显示多个图形，例如：

>>plot(Y);

>>plot(X1,Y1,X2,Y2,...);

当绘图指令不正确时，SIMULINK将在模块图标中以(? ? ?)标示出来。

（3）图像的显示

封装编辑器中的image 和patch功能可以让用户在封装模块图标中显示位图或绘制小图片。

image(a)指令显示由a定义的图像，其中a是由RGB值组成的M×N×3数组。可以用指令imread和ind2rgb将windows的图形文件转换成必要的矩阵格式，例如：

>>image(imread('icon.tif'))

其结果是在MATLAB路径当中读入名为icon.tif的TIFF图形文件：

>>image(a, [x, y, w, h])

其结果是在相对封装模块左下角，坐标为[x, y]的位置显示图像，图像的高度和宽度分别为h和w。

patch(x, y)创建一个实心的图形，它的形状由x 和y向量的坐标决定。其颜色是当前的背景颜色。如：

>>patch([0 .5 1], [0 1 0], [1 0 0])

创建一个红色三角形的封装图标。

（4）在模块图标中显示传递函数

为了在模块图标中显示传递函数，可以在画图命令域中输入下列的指令：

>>dpoly(num, den)

>>dpoly(num, den, 'character')

其中，参数num 和 den分别为传递函数分子和分母的多项式系数，一般在初始化命令当中定义。传递函数的字符由参数character指定，缺省时为s。当封装模块绘制图标时，SIMULINK将执行初始化命令，然后在图标中将传递函数显示出来。如果要以s的降幂次序显示连续系统的传递函数，可以输入：

>>dpoly(num, den)

>>dpoly(num, den, 'z')

例如，输入：

>>num = [0 0 1];

>>den = [1 2 1];

图标将如图 7.8（a）所示。

如果以1/z的升幂次序显示离散系统的传递函数，可以输入：

>>dpoly(num, den, 'z-')

例如，对于上例同样的参数num 和den ，图标如图7.8（b）所示。

如果要以零极点方式显示传递函数，可以输入：

>>droots(z, p, k)

例如，对于下列的零极点参数，最终的图标如图7.8（c）所示。

>>z = []; p = [-1 -1]; k = 1;

如果上述指令中的参数没有定义，或者没有具体数值，SIMULINK将在图标中显示（???）。这些参数还可以通过封装对话框中由用户自己输入。

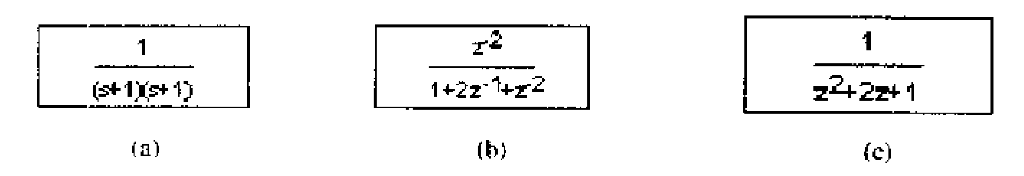


图 7.8　在图标中显示传递函数

（5）控制图标属性

用户还可以通过画图命令域下面的选择项来控制封装模块的图标属性。

图标框架：图标框架是出现在模块四周的矩阵，我们可以设置图标框架（Icon frame）为Visible或Invisible来控制图标框架的可见与不可见。缺省情况下图标框架总是可见的。

透明图标：图标可以被设置成透明的，也可以以不透明方式进行显示。缺省情况下的显示方式为不透明，这时，被图标覆盖的内容将隐藏起来。

图标旋转：当模块进行旋转时，我们可以设置模块的图标是否跟着进行旋转。缺省情况下，图标是固定的。图标旋转时的方向与端口的旋转方向一致。图显示的是图标进行旋转时的情况

画图坐标：这个参数控制绘图命令所使用的坐标系统。其值可以为Autoscale, NormalizedandPixel之一。

3. 文档属性页（Document page）的设置

文档属性页可以让用户在其中修改封装模块的类型，定义描述文本和帮助信息，如图 7.4 所示。各部分的含义介绍如下：

（1）封装类型域

封装类型仅仅是对封装模块的一种分类信息。它出现在该模块的模块对话框和所有的封装编辑器中。我们可以在其中输入需要的信息。SIMULINK 创建封装对话框时在封装类型后加上"（mask）"，以便同标准的模块进行区分。

（2）模块描述域

模块描述是一些将来出现在封装模块对话框中封装类型下面的几行描述性文字，其中一

般是对该封装模块主要功能的简单介绍。SIMULINK 可以根据需要自动为文字分行，当然我们也可通过回车键强制分行。

（3）封装帮助文本域

用户可以在该域中输入封装模块的相关帮助信息，例如模块的使用方法，参数的含义以及其他注意事项。当使用者在模块对话框中按下 Help 按钮时，这些帮助信息就会告诉使用者该封装模块的相关情况。这为封装模块面向第三方的发布提供了便利。用户还可以在其中使用 URL 资源，让使用者通过 Web 浏览器获取相关帮助，具体设置方法可以参考有关的 PDF 文档。

4. 封装模块动态对话框的创建

动态对话框是指对话框的状态能够依照用户的输入动态变化的对话框。在 SIMULINK 中，我们可以为所封装的模块实际动态对话框，使封装模块的使用更加方便友好，动态变化的内容包括参数控制的可见性，使能状态以及参数值的缺省定义等等。

创建动态对话框的具体过程可通过封装编辑器和 set_param 指令共同完成。我们可以首先用封装编辑器定义对话框将出现的所有参数，无论是动态变化的还是静止的。然后通过在命令窗口中通过输入适当的 set_param 指令定义相应的回调函数（callback functions），该函数可以依照用户的输入进行相应的动作。最后通过模块保存操作将动态封装对话框的有关设置记录下来。其中回调函数的设置方法可以参见有关的帮助文档。

7.4　条件子系统

7.4.1　使能子系统

使能子系统在仿真过程的每一步都将检查控制信号的符号，只有当控制信号为正时，该系统才发生作用。这里的控制信号可以为标量，也可以是矢量。如果是标量，则该只要该标量的信号值比零大，则该子系统被执行。如果是矢量，则只要其中有一个分量大于零，就满足子系统执行的条件。

例如，假设控制输入是一个正弦波信号，则子系统的开启和关闭交替发生，如图 7.9 所示。其中向上的箭头表示子系统有效，而向下的箭头表示子系统无效。

可以通过从模块库中拷贝一个使能模块到子系统窗口中来建立使能子系统，如图 7.10 所示。

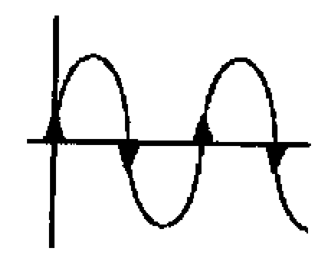


图 7.9　正弦波的控制输入信号

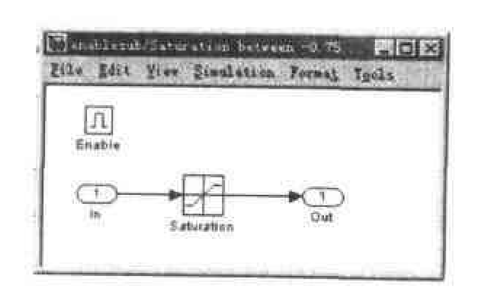


图 7.10　向子系统中拷贝使能模块

这里有几个问题需要注意：

一是虽然使能子系统在外界条件不满足时不会被执行，但不是说它的输出就变得无效了，我们仍然需要为该系统指定输出。当该子系统被禁止时，我们可以设置它的输出保持前一步仿真时的值，也可以将它设置成初始状态值。设置过程是在使能子系统的模块对话框中完成的。双击使能模块的图标，系统将弹出图所示的模块对话框，为 Output when disabled 项指定 held 或 reset 之一。其中，held 表示子系统被禁止时的输出保持前一步仿真时的值，而 reset 表示子系统被禁止时的输出设置成初始状态值，这里的初始状态值是在 Initial output 中输入的。

二是需要设定子系统再次执行时的状态。我们同样可以设定子系统再次启动时的状态为前一步仿真时的值，也可以重置为初始状态，这可以通过为 States when enabling 项设定成 held 或 reset 之一来实现，如图 7.11 所示。

三是控制信号的输出问题。在使能模块对话框中有一项 Show output port 的选择框，如图 7.12 所示，选中它可使控制信号输入到子系统内部，这在子系统内部需要控制信号的情况下特别有用。

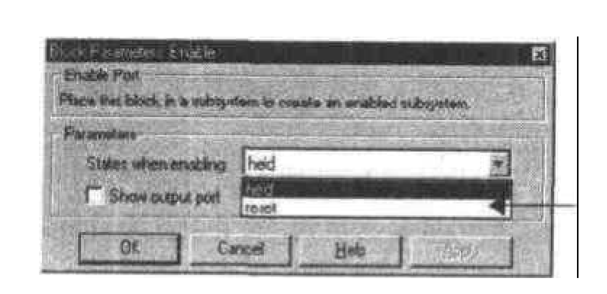


图 7.11　设定子系统再次执行时的状态

图 7.12　不同采样频率的离散使能子系统

在使能子系统内部，我们可以加入各种模块，并用信号线连接起来，这样就完成了使能子系统的创建。使能子系统既支持连续系统，也支持离散系统，下面我们讨论一下具有不同采样频率的离散使能子系统的相关问题。

假设我们创建了如图 7.11 所示的离散使能子系统，要讨论模块包括：

模块 A：采样周期为 0.25 秒；

模块 B：采样周期为 0.5 秒；

模块 C：在使能子系统内部，采样周期为 0.125 秒；

模块 D：在使能子系统内部，采样周期为 0.25 秒。

控制信号由脉冲发生器产生，标记为 Signal E，以上四个模块的执行情况如图 7.11 所示。向上的箭头表示相应的模块发生作用。其中，模块 A、B 由于不在使能子系统内部，因此不受控制信号的约束。而模块 C、D 在控制信号为正时在才保持正常的采样周期。

7.4.2　触发子系统

触发子系统外观上有一个触发控制信号输入口。每当触发事件发生时，系统就接受输入端的信号。触发事件由子系统内的触发模块对话框定义。有四种触发事件形式选择：

rising 上升沿触发　　触发信号以增长方式穿越 0 时，触发事件发生。

falling 下降沿触发　触发信号以下降方式穿越 0 时，触发事件发生。

eighter 任意沿触发　每当触发信号穿越 0 时，触发事件都会发生。

function-call 函数调用触发这种触发方式必须与 S 函数配合使用。

SIMULINK 采用不同的图标来表示不同触发方式的触发模块或子系统。

图 7.14 显示的是某个触发子系统的例子，在这个例子当中采用的是上升沿的触发方式。

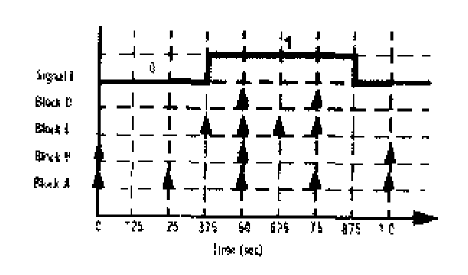


图 7.13　不同采样模块的采样时刻示意图

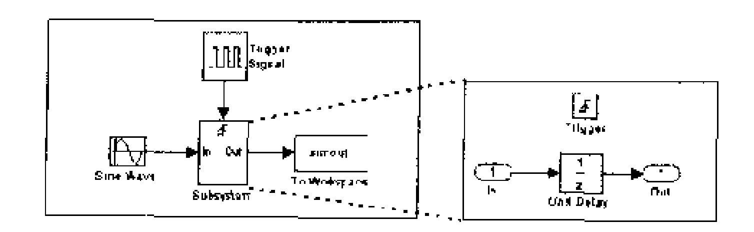


图 7.14　采用上升沿触发方式的触发子系统

可以通过从模块库中拷贝一个触发模块到子系统窗口中来建立触发子系统。为了设定相应的触发类型，在触发模块对话框中将 Trigger type 项设定成相应的值。

下面讨论几个值得注意的问题：

一是触发事件之间系统的输出和状态。与使能子系统不同，触发子系统在两个触发事件之间无论是输出信号还是内部状态都保持最新的变化值不变。

二是触发信号的输出问题。通过选中 Show output port 选项，我们同样可以将触发信号输出到子系统内部，下面的 Output data type 选项表明触发信号的数据格式，如果设置成 auto（自动）形式，则触发信号会根据所连接的信号类型自动转化为 8 位整数形式或双精度形式。

三是不是所有的模块都可以放置在触发子系统中。能够放置的模块有：对采样操作具有“继承”特性的逻辑模块和增益模块；采样时间被设置成-1 的离散模块。一般的连续时间模块不能用于触发子系统。

7.4.3　触发-使能子系统

我们可以把触发模块和使能模块放置在同一个子系统中，这样的系统称之为触发使能子系统。该系统同时具有触发信号输入和使能信号输入，其行为方式与触发子系统相类似，但只有当使能信号为正时，触发事件才起作用。

图 7.15 显示的是一个简单的触发使能子系统，观察其中的子系统图标，看看与前面的系统有何不同。

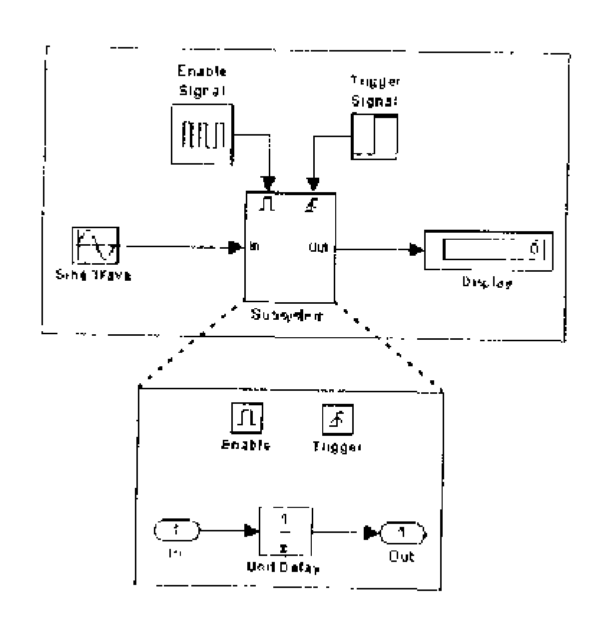


图 7.15　触发使能子系统

7.4.4　交替执行子系统

条件执行子系统与 SIMULINK 模块库中的 merge 模块配合使用，可以构成所谓的交替

执行子系统。例如，图 7.16 显示了一个简单的交替执行子系统，其功能是将信号取绝对值，即正的信号保持不便，负的信号变成正的信号。图 7.17 显示了图所示系统的仿真结果。

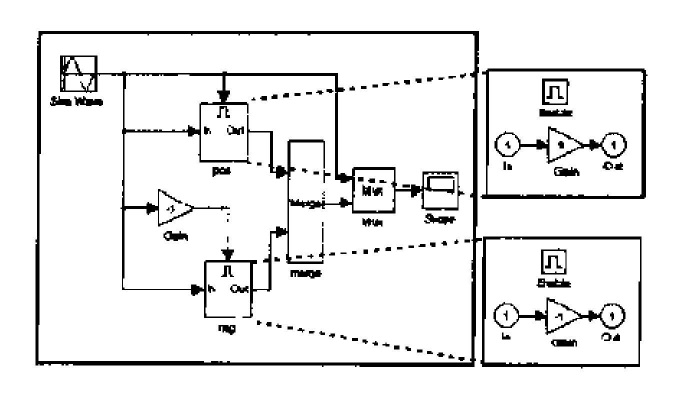


图 7.16　交替执行子系统

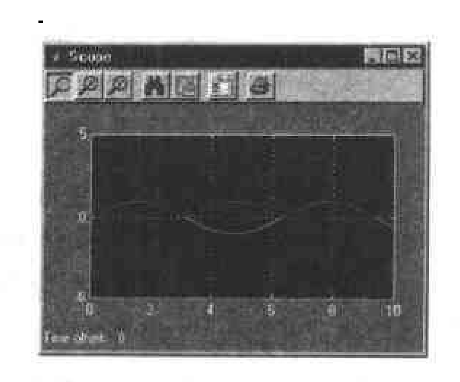


图 7.17　交替执行子系统的仿真结果

7.4.5　条件子系统小结

在实际建模过程中可以发现，有的条件子系统完全可以采用其他的基本模块如 Switch 或 Relational operator 来实现。但是两者的不同点在于，如果直接采用基本模块构成模型，那么在仿真过程中就必须时刻对模型中对每个模块进行计算，而不论这时的模块对系统是否有影响，这将造成计算资源的浪费，而采用条件子系统则可以避免这种情况，因为条件子系统在外部条件不满足时，不会计算该子系统中的模块。

在许多实际系统中，我们经常会遇到多种模式运行的系统，如飞机的平飞状态与着陆时是不同的，在许多理论研究中，也会经常遇到用不同算法处理不同情况和条件的问题，这时，条件子系统为我们创建相应的仿真模型提供了极大的方便。

第 8 章　仿真模型的分析

用户在创建 SIMULINK 仿真模型之后，在启动仿真过程之前一般需要对所创建的模型进行分析，其目的是为了以后正确设置仿真的参数和配置。例如所创建的模型究竟存在多少状态，是否存在代数环，采用哪种微分方程求解器，积分步长和容许误差究竟取多大合适等等，这些问题对于提高系统模型的仿真质量（速度和精度）具有相当重要的意义，其中有些问题是初学者常常容易忽视而又经常遇到的，因此有必要将这些问题单独组织在一起进行专门论述。

8.1　模型状态的确定

8.1.1　导引

在进行 SIMULINK 仿真过程中常常需要为仿真模型设置初始状态，然而对于实际比较复杂的系统模型，一般很难直接看出模型状态的个数以及相关变量。尤其当用户在命令窗口中运行仿真过程时，就必须事先知道系统模型当中究竟有多少个状态，其中多少是连续量，多少是离散量，模型中的模块对应着哪一个状态分量。

从本质上而言，各种模块就是图形化的微分或差分方程。无论是高阶微分或差分方程，还是传递函数，SIMULINK 在其内部总是采用连续或离散状态方程加以描述。

对于简单的系统，可以直接根据模块的特点分析出模型的状态：积分模块直接对应着一个连续状态变量；单位延迟模块对应着一个离散状态，而传递函数和状态空间模块所对应的状态变量数目由模块本身的阶次决定。

对于不能直接看出的系统，可以在命令窗口中输入相应的指令进行分析。

8.1.2　确定模型状态

从模型获取状态信息的指令就是模型名称本身。具体格式是：

>>[sizes，x0，statecell] = model

其中，model 为具体模型名。Sizes 输出参数是必须的，它是一个 7 元向量，其中各部分的含义如下：

size(1)：状态向量中连续分量的数目

size(2)：状态向量中离散分量的数目

size(3)：输出分量的总数

size(4)：输入分量的总数

size(5)：系统中不连续解的数目

size(6): 系统中是否含有直通回路

size(7): 不同采样速率的类别数

从上面可以看出，数组 sizes 包含该模块的许多基本特征。

x0 返回的是模型状态向量的初始值，可以通过对 x0 赋值来设置状态向量的初始值。

Statecell 按次序给出了所有状态变量对应模块的所在模型名称、子系统名以及模块名。

8.1.3　平衡点的确定

在非线性系统分析中，分析系统稳定性或稳定过程的动态特性时经常是在系统的平衡点处进行的，因此如何从仿真模型中获取平衡点就显得比较重要了。

所谓平衡点（Equilibrium point）是指所有状态导数等于零的点。如果只有部分状态导数等于零，则称为偏平衡点。

SIMULINK 提供的 trim 指令可以用来确定稳态平衡点，它的工作原理是采用最优化方法，从一个初始值开始搜索一个使状态导数的最大绝对值最小化。最终搜索的结果并不保证搜索到的平衡点一定收敛，也不保证其结果是最优的，实际上，最终的结果与初始值选取是否得当有关，如果没有得到收敛值，可换其他的初始值进行搜索。

trim 指令的调用格式为：

```
>>[x,u,y,dx,option]=trim('model',x0,u0,y0,ix,iu,iy,dx0,idx,option,t)
```

x0,u0,y0 是开始搜索时，(x,u,y)，即状态、输入和输出的初始值。它们都采用列向量的形式。

Ix,iu,iy 分别用来指定 x0,u0,y0 中保持不变的分量下标，使平衡点的搜索过程受约束进行。

dx0 与 idx 配合使用。Idx 指定哪些状态分量的导数不等于零，dx0 指定非零导数的具体值。

option 使优化算法的参数选项设置。

t 是时变状态导数的具体计算时刻。

8.1.4　实例

这一部分通过两个例子来说明模型特征的获取方法。

例8.1 以MATLAB的演示模型vdp.mdl为对象，确定模型的状态。

```
>>[sizes, x0, statecell] = vdp
sizes =
       2
       0
       2
       0
       0
       0
       1
x0 =
       2
       0
statecell =
    'vdp/x1'
        'vdp/x2'
```

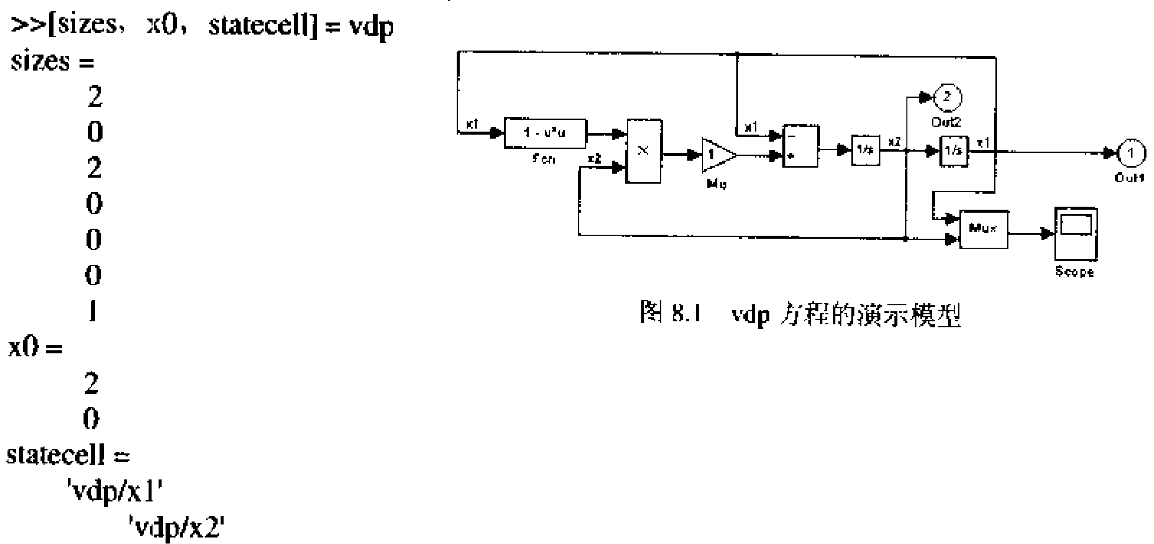


图 8.1　vdp 方程的演示模型

结果显示，该模型中的积分模块 x1、x2 形成了惟一的两个连续状态。

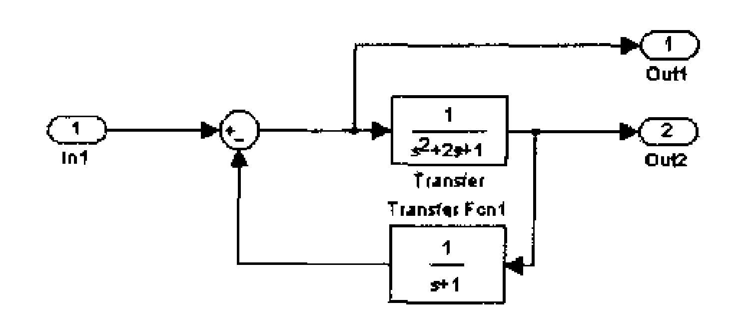


图 8.2　连续系统模型

例8.2 如图8.2所示的模型，确定模型的平衡点。

```
>>x = [0; 0; 0]; u = 0; y = [1; 1];
>>ix = [ ];
>>iu = [ ];
>>iy = [1; 2];
>>[ x, u, y, dx] = trim('model71', x, u, y, ix, iu, iy)
x =
     1.0000
          0
     1.0000
u =
     2.0000
y =
     1.0000
     1.0000
dx =
      0
      0
      0
```

结果显示系统的平衡状态位于(1, 0, 1)处。

8.2　模型的线性化问题

8.2.1　线性化的数学描述

在进行非线性系统的仿真研究中，常常需要将非线性系统在工作点附近进行线性化近似，然后再采用成熟的线性分析方法来研究非线性系统在工作点附近的一些动态特性。这一节我们将讨论如何利用 SIMULINK 获取非线性系统的近似线性模型。

非线性系统可以通过如下的状态方程进行描述：

$$\begin{cases} \dot{x} = f(x,u,t) \\ y = g(x,u,t) \end{cases} \tag{8.1}$$

式中，$\dot{x}$，f 都是 n 阶向量，u 是 m 阶向量；y 是 p 阶向量。在系统工作点附近进行近似线性化，只保留系统的一次项，而将系统的高次项都舍去，可以得到如下的方程。

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \tag{8.2}$$

其中，各参数矩阵的计算如下：

$$A = \frac{\partial}{\partial x} f(x,u,t)$$
$$B = \frac{\partial}{\partial u} f(x,u,t)$$
$$C = \frac{\partial}{\partial x} g(x,u,t)$$
$$D = \frac{\partial}{\partial u} g(x,u,t) \tag{8.3}$$

这里的近似只是相对比较简单的系统在平衡点附近而言，对于非线性特性较为复杂的系统，这样的近似可能会引起系统特性的较大变化。

8.2.2　连续系统的线性化

采用 linmod 或 linmod2 指令可以进行连续非线性系统的近似线性化，二者的调用格式基本类似，但 linmod2 比 linmod 运行需要更长的时间，产生的结果要比 linmod 精确。具体调用方法为：

```
>>[A, B, C, D]=linmod('model')
>>[A, B, C, D]=linmod('model', x, u)
>>[A, B, C, D]=linmod('model', x, u, para)
>>[A, B, C, D]=linmod2('model')
>>[A, B, C, D]=linmod2('model', x, u)
>>[A, B, C, D]=linmod2('model', x, u, para)
>>[A, B, C, D]=linmod2('model', x, u, para, Apert, Bpert, Cpert, Dpert)
```

其中，x、u 用来指定工作点处的系统状态和输入信号。缺省时均为适当的零向量。

model 为具体的模型名称。

para 是一个二维向量。para(1)指定计算数值偏导数时使用的扰动值。对于 linmod，缺省值为 10^{-5}，对于 linmod2，其缺省值为 10^{-8}。para(2)指定工作点处的时间，缺省时为 0。

xpert，upert 分别指定各状态分量、输入分量的扰动值。

Apert，Bpert，Cpert，Dpert 分别为与各元素联系的状态和输入的扰动值。我们可以输入

```
>>[A, B, C, D, Apert, Bpert, Cpert, Dpert]=linmod2('model', x, u, para)
```

来获取它们的实际值。

linmod 得到的模型是以状态空间 A、B、C、D 形式来表示其输入和输出之间的关系。它的工作过程是在工作点附近对状态施加扰动后来确定状态导数和雅可比矩阵，并把得到的结果用来计算状态空间矩阵。每一个状态 $x(i)$ 加上扰动后变成：

$$x(i) + \Delta(i) \tag{8.4}$$

其中：

$$\Delta(i)=\delta(1+|x(i)|) \tag{8.5}$$

同样，第 j 输入受到扰动后，变成：

$$u(i)+\Delta(i) \tag{8.6}$$

其中：

$$\Delta(i)=\delta(1+|u(i)|) \tag{8.7}$$

8.2.3 离散系统的线性化

离散系统的线性化可采用 dlinmod 指令，具体用法如下：

>>[A，B，C，D]=dlinmod(‘model’，Ts，x，u，para)

Ts 指定了近似线性模型的采样时间。如果 Ts=0，则得到的是与原系统近似的连续模型。

x，u 用来指定工作点处的系统状态和输入信号。

para 是一个二维向量。para(1)指定计算数值偏导数时使用的扰动值，缺省值为 10^{-5}，para(2)指定系统工作点处的时间，缺省时为 0。

如果一个模型是由线性模块、多速率模块、离散模块和连续模块组成的，那么在满足下列条件的情况下，dlinmod 产生一个采样时间为 Ts，并且所得线性模型在 Ts 采样点上与原系统有相同的频率响应和时间响应的线性系统。

（1）原系统稳定；

（2）Ts 不小于最慢的采样周期；

（3）Ts 为原系统所有采样周期的整数倍。

由于离散系统的稳定性与采样时间有关，因此进行近似线性化以后系统的稳定性有可能发生变化，这可以通过观察输出矩阵 A 和采样时间 Ts 进行判断。

在 Ts=0 时，如果 A 的所有特征值均具有负实部，或者在 Ts>0 时，如果 A 的特征值在单位圆内，则系统稳定。

当系统不稳定且采样时间不是其他采样时间整数倍的时候，dlinmod 就可能产生复矩阵 Ad 和 B。然而在这种情况下，仍然可以通过矩阵 A 的特征值来验证系统的稳定性。

dlinmod 可以用来把一个系统的采样时间变成其他值，或者把一个线性离散系统变成一个连续系统，或者把一个连续系统变成一个离散系统。

8.2.4 实例

下面通过一个连续系统的例子来说明求近似线性系统的方法。

例8.3 同样研究图7.1的模型，求它在坐标原点处的线性化模型，并分析原点处的稳定性。

（1）求模型在坐标原点处的线性化模型

```
>>[A, B, C, D]=linmod('vdp');  A
A =
 0     1
-1     1
```

（2）求模型在坐标(2,4)处的线性化模型

```
>>[A1, B1, C1, D1]=linmod('vdp', [2, 4]);   A1
A1 =
     0      1.0000
  -17.0000   -3.0000
```

（3）分析模型原点处的稳定性

```
>>Ae=eig(A), A1e = eig(A1)
Ae =
  0.5000 + 0.8660i
  0.5000 - 0.8660i
A1e =
  -1.5000 + 3.8406i
  -1.5000 - 3.8406i
```

由计算结果可知，在原点处的模型是不稳定的，而在（2,4）处的模型是稳定的。

8.3　代数环问题

8.3.1　仿真模型中的代数环

SIMULINK 当中有一些模块具有“直通”特性的输入端口，这意味着如果不知道模块的输入信号就无法求取它的输出信号，这些模块我们称作直通模块（Direct Feedthought Block）。直通模块从时间关系上讲具有无记忆、无延迟的特征，模块的输入输出之间只包含代数关系。常见的直通模块有：

增益模块（Gain）
乘运算模块（Product）
求和模块（Sum）
基本数学模块（Abs,Sign,Logical,Math Function）
非线性模块（Saturation,Quantizer,Relay）
D 矩阵非零的状态方程模块（State-Space）
分子分母同阶的传递函数模块（Transfer Fcn）
零极点数目相同的零极点增益模块（Zero-Pole）
积分模块的初始条件输入端口（Integrator）

如果在模型当中存在全部由直通模块组成的环形通路，则我们说这个模型具有代数环（algebraic loop），这个环形通路就是代数环本身。图 8.3 显示的是由求和模块构成的一种最简单的代数环。

从数学上讲，这个求和模块可以用方程 z=u-z 来表示，很容易看出，这个简单代数环的解为 z=u/2，但是绝大多数代数环都无法通过直接观察就得到它的解析解，更多的情况是采用多个变量来表示本来是相同的变量，即 z1,z2。就本例来说，SIMULINK 将它转换成图 8.4 所示的模型。

代数环给系统带来的直接危害是影响仿真计算的速度。因为 SIMULINK 在处理模型中的

代数环时，并不是像我们手工那样将相同的项合并（一般不可能机器实现），而是采用其他类似迭代的方法加以解决。这种迭代在每一个积分步长的计算中都要进行，因此增加了仿真计算的额外消耗。另外，尽管采用的迭代方法具有一定的鲁棒型，但如果初始值选取不合适，最终的结果不能保证是收敛的。

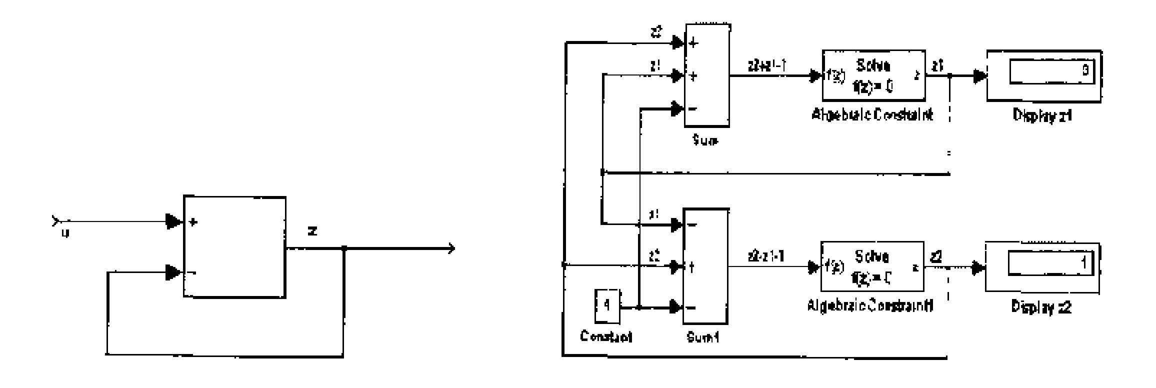


图 8.3　由求和模块构成的最简单的代数环　　　　图 8.4　代数环的实际求解等效模型

因此我们应当尽量避免仿真模型中的代数环。当然，如果代数环所引起的计算速度变慢的问题可以容忍的话，我们也不必过于介意，如果代数环不可避免，可以通过增加记忆功能模块和代数约束模块消除模型中的代数环。

8.3.2　非代数环的情况

不是所有的由直通模块组成的环都是代数环，如果：

- 环中存在触发子系统；
- 该环是从输出到某个积分器的重置端。

对于存在触发子系统的情况，求解器可以假定触发子系统的输入在触发瞬间是不变的，这就可以根据前一时刻的输出来计算当前时刻的输入，从而免去了求解代数环的需要。

例 8.4　如图 8.5 所示的仿真模型，该系统可以用 z=1+u 来表示。u 是触发子系统上一触发时刻的 z 值，系统的输出呈现出图显示的阶梯信号。如果我们删除系统中的触发信号，如图 8.6 所示，这样系统就表示为 z=1+z，该代数环无解。

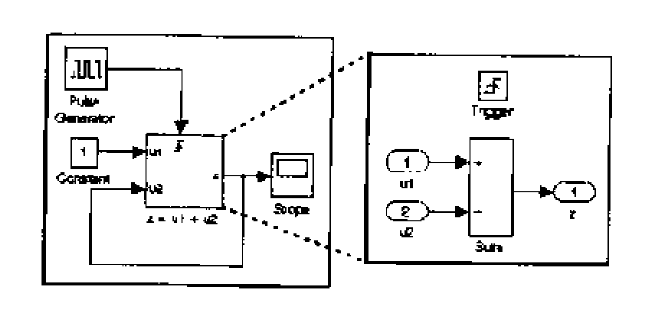


图 8.5　包含代数环的触发系统

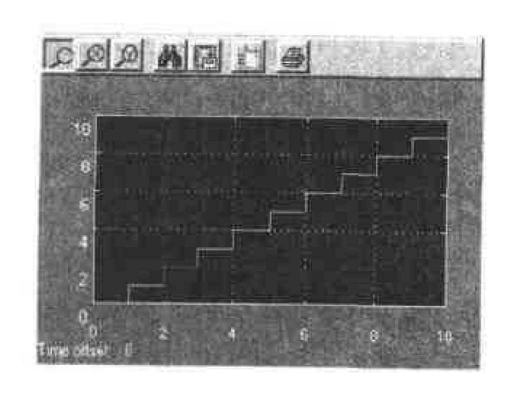


图 8.6　包含代数环的触发系统的仿真曲线

8.4 微分方程的求解算法

8.4.1 微分方程的求解

我们知道，目前数学上存在多种求解微分方程的算法，有的算法是很经典的。但是在众多的求解算法中，一般不存在通用的、最好的求解方法，具体运用那种方法需要具体情况具体分析，这就需要我们在选择哪种求解方法之前仔细分析系统的动态性能。

SIMULINK 提供了多种求解微分方程的求解方法，如 Runge-Kutta 法、Adams 法、Gear 法、Euler 法、linsim 法等等。SIMULINK 4.0 中还增加了其他包括求解两点边值问题的算法。

简单的讲，Runge-Kutta 法适合于高度非线性或不连续的系统，不适合于刚性系统（既有快变特性又有慢变特性的系统）；Adams 法适合于非线性不大，时间参数变化小的系统；Gear 法专门用于刚性系统的仿真，对非线性系统的仿真能力较差。Euler 方法比较差，一般不采用；linsim 法适合于近似线性的系统，尤其适合于线性刚性系统。

8.4.2 各种求解方法的比较

下面针对 SIMULINK 中经常使用的几种求解算法进行分析。

1. ODE45 和 ODE23

这两种方法都属于 Runge-Kutta 法，都是用有限的 Taylor 级数去近似解函数。有限项 Taylor 级数近似的主要误差是所谓的阶段误差（由截去的高阶项引起）。

ODE45 分别采用四阶和五阶 Taylor 级数计算每个积分步长最后的状态变量近似值，并把这两个近似值的差作为对阶段误差大小的估计。如果误差估计值比较大，那么就把该积分步长缩短，然后再重新进行计算，直到误差估计值小于指定的精度范围。如果误差估计值远小于指定精度，那么将把下一个积分步长缩短。

ODE23 与 ODE45 的区别在于：在每一个积分步长中，是采用二阶和三阶 Taylor 级数计算每个积分步长最后的状态变量近似值的。

上面的论述表明 ODE23 和 ODE45 都是可变步长的。

一般而言我们在进行仿真参数设置时，首先选择 ODE45 方法。在规定同样的精度条件下，ODE23 的积分步长要比 ODE45 方法小。

2. ODE113

ODE113 是变阶的 Adams 方法，属于一种多步预报校正算法。

在预报阶段，ODE113 采用阶多项式近似导函数，多项式的系数可以通过前面 n-1 个解及其导数值确定；然后采用外推方法计算下一个解。在校正阶段，通过对前面 n 个解和新得到的解，通过拟合或得校正多项式，最后用校正多项式重新计算解，从而对原有解进行校正。在预测解和校正解之间的误差可以用来调整积分步长。

由于 ODE113 采用多项式近似，因此可以有效求解系统微分方程，不过一般不用来计算

含间断点的系统；与 ODE23 和 ODE45 比较，ODE113 计算导数得次数相对少一些。

3. ODE15S 和 ODE23S

ODE15S 一般用来求解刚性（stiff）系统，属于一种变阶多步算法，名称后面的“s”即表示专门用于刚性系统的求解。所谓刚性系统，是指系统特征值相对距离较远的系统，因此一般既包含较快的动态特性，又有很慢的动态模式。ODE15s 算法当中包含一种对系统这种不同的动态特性进行检测的机制，但它对非刚性系统的计算效率不高，尤其是对于包含快速变化特性的系统。

ODE23s 同样是针对刚性系统而言的，属于定阶单步算法。

8.5　积分步长与容许误差

8.5.1　积分步长的选择

积分步长包括初始步长、固定步长、最大步长等不同情况，设置时需要根据系统的特性和所采用的求解方法来选择。

最大步长（Max step size）的设置：针对变步长算法而言，一般设置成自动（auto）方式，这时最大步长大约是仿真时间的 1/50，如果不是特殊情况，最大步长设置成缺省方式（auto）就可以完全满足要求。如果仿真时间很长，也可以设为其他值，如果是周期系统，则可以定为周期的 1/4。值得注意的是，如果最大步长选择过大，在仿真计算时可能会跳过系统的某些特性，因此如果需要了解某段时间内系统动态特性的细节，不妨将该值调小些。

初始步长（Initial step size）的选择：针对变步长求解算法而言，缺省时为自动方式（auto），这时系统会根据初始状态导数的大小计算初始步长。如果初始步长选择过大，可能会忽略系统启动时的某些性状，因此如果所研究的系统在启动瞬间变化剧烈，应该把该值取小一些。注意，这里的初始步长并不是实际的计算步长，而是系统第一次选择的步长，如果根据该步长计算的结果超过了系统的容许误差，系统将自动减少步长重新进行计算。

固定步长（Fixed step size）的设置：针对固定步长的算法而言。对于离散系统或混合系统，如果选用定步长方式进行计算，那么有三个工作方式进行选择：

（1）单任务（Single Tasking）方式：主要用于“单任务”系统。运行中不检查模块间是否存在速率突变。

（2）多任务（Multi Tasking）方式：用于多速率或混合系统，运行时检查各模块的工作速率有无冲突。

（3）自动（Auto）方式：定步长算法的缺省工作方式，对系统的检测自动进行。如果整个系统采用同一采样速率，则以单任务方式运行，否则以多任务方式运行。

8.5.2　容许误差的设置

SIMULINK 当中的微分方程求解器使用标准的局部误差控制算法来监测每一步仿真中的计算误差。在每个仿真步中，求解器计算所有的状态值，同时计算该步仿真中误差的大小，

并同设置的容许误差进行比较，如果计算误差超过了容许误差的限制，则取消这步仿真步骤，并减小仿真步长重新计算。容许误差是由相对容差（rtol）和绝对容差（atol）共同决定的，即 $e_i \leq \max(\text{rtol} \times |x_i|, \text{atol}_i)$。当 x_i 本身绝对值较大时，计算精度主要由相对容差进行控制；当 x_i 本身绝对值较小时，计算精度主要由绝对容差进行控制。

一般情况下，用户只需要为容许误差指定缺省值。如果用户指定了缺省方式，则 SIMULINK 指定绝对容差的初始值为 10^{-6}，之后绝对容差将随着仿真过程进行调整。如果容许误差的缺省值不合适，则可以人为指定其他值。不过，用户需要多次运行仿真才能最终确定合适的容许误差值。如果需要为不同的状态指定分别指定不同的容许误差，可以在模型中相应的积分模块中进行设置。

第9章　运行仿真

用户在创建系统的仿真框图并对它进行分析之后，就可以启动仿真过程了。SIMULINK支持两种不同启动仿真的方法：直接从模型窗口中启动和在命令窗口中启动。无论哪种方法，最终的仿真过程是相同的。在仿真启动之前，用户还需要仔细配置仿真的基本设置。如果有的设置不合理，仿真过程甚至可能无法进行下去。

本章将讲述运行仿真之前的一些准备工作，包括仿真具体配置的实现，和运行仿真的一般步骤、结果的显示，以及如何提高仿真速度和精度的一些措施等等，旨在使读者了解SIMULINK 仿真的基本过程，学会 SIMULINK 仿真的基本操作。

9.1　启动仿真过程

9.1.1　仿真入门

有两种方法启动仿真过程，一种是在命令窗口中以指令形式开始相应模型的仿真，另一种是直接在模型窗口菜单中单击相应的菜单命令。

1. 菜单方式

采用菜单命令形式启动仿真过程是相当简单和方便的。用户可以在配置对话框中完成诸如仿真起止时间、微分方程求解器、最大仿真步长等等参数的设置，而不需要去记忆相关指令的用法，尤其是用户可以在不停止仿真过程的情况下实时地完成下面的操作：

- 修改仿真起止时间、最大仿真步长。
- 改变微分方程求解方法。
- 同时进行另一个模型的仿真。
- 用鼠标单击相应的信号线，在出现的浮动窗口中观看信号的相关信息。
- 在一定范围内修改模块的某些参数。

注意，在仿真过程中用户不能再改变模型本身的结构，如增减信号线或模块，除非停止该模型的仿真过程。

2. 命令行方式

相比菜单方式，采用命令行方式启动仿真过程具有如下的优点：

- 仿真的对象既可是 SIMULINK 方框图，也可以是 M 文件或 C MEX 文件形式的模型。
- 可以在 M 文件中编写仿真指令，从而可以实时改变模块参数和仿真环境。

9.1.2 用菜单方式启动仿真

用菜单方式启动仿真一般分为以下几个步骤:

（1）设置仿真参数

单击菜单项“Simulation: Parameters”，弹出仿真配置对话框。一旦用户按下“Apply”或“Close”按钮，即表示目前的设置生效。关于如何进行相关参数的配置，下面我们还会具体论述。

（2）开始仿真

单击菜单项“Simulation：Start”即可启动仿真，当然用户还可以采用“Ctrl-T”的快捷键方式。仿真结束后，计算机将发出“哔”的声音来提示用户。

> 说明：这里用户常遇到的错误是在 SIMULINK 模块库窗口激活状态下启动某个模型的仿真过程。为避免这种错误，用户在启动仿真时需要确信待仿真的模型窗口当前是激活的。

由于模型的复杂程度和仿真时间跨度的大小不同，每个模型的实际仿真时间不相同，同时仿真时间还受到机器本身性能的影响。用户可以在仿真过程中单击“Simulation：Stop”菜单项，或采用【Ctrl+T】的快捷键方式人为中止模型的仿真。用户也可以选择“Simulation：Pause”或“Simulation：Continue”菜单项来暂停或继续仿真过程。如果模型中包含向数据文件或工作空间输出结果的模块或在仿真配置中进行了相关的设置，则仿真过程结束或暂停后会将结果写入数据文件或工作空间中。

9.1.3 仿真过程的诊断

如果仿真过程中出现错误，仿真将会自动停止，并弹出一个仿真诊断（Simulation Diagnostics）对话框来显示错误的相关消息，如图 9.1 所示。

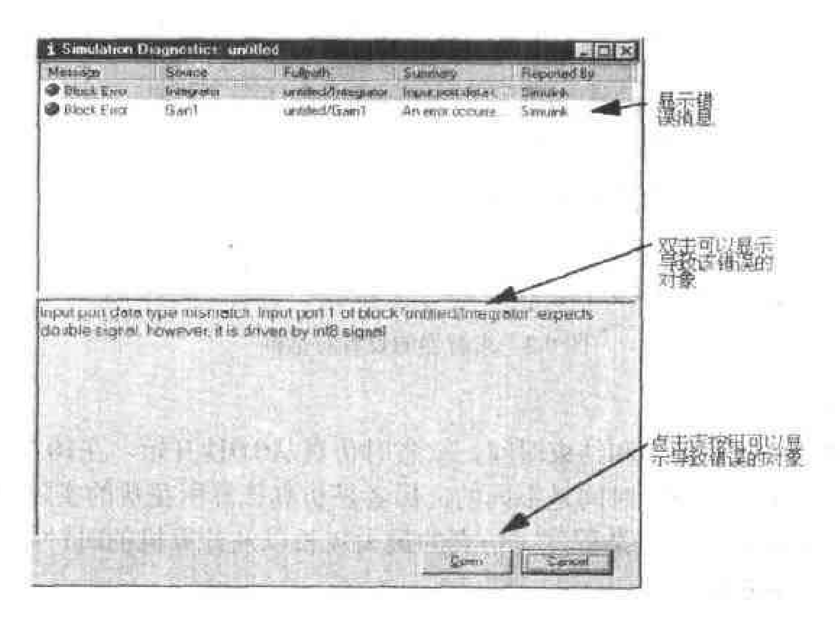


图 9.1　SIMULINK 的错误消息对话框

该对话框分为上下两个部分，上面的部分是每个错误消息的基本情况，下面部分是对相

应错误的详细描述。仿真诊断对话框将显示错误的如下信息：

Message.	错误类型，诸如模块错误或警告等等
Source.	发生错误的模块名称。
Fullpath.	导致错误的对象的完整路径
Summary	错误的简单说明
Reported by	报告错误的组件，如是 SIMULINK、Stateflow 还是 Real-Time Workshop 检测到该错误。

SIMULINK 除了弹出诊断对话框显示错误信息外，必要时还会弹出模型方框图，并采用高亮方式显示引发该项错误的相关模块，用户也可通过在诊断对话框中双击相关的错误或按下“Open”按钮来弹出相应的方框图。

9.2　仿真的配置

9.2.1　求解器的设置

求解器属性页如图 9.2 所示，其功能是完成仿真起止时间、求解器类型以及输出选项的设置。

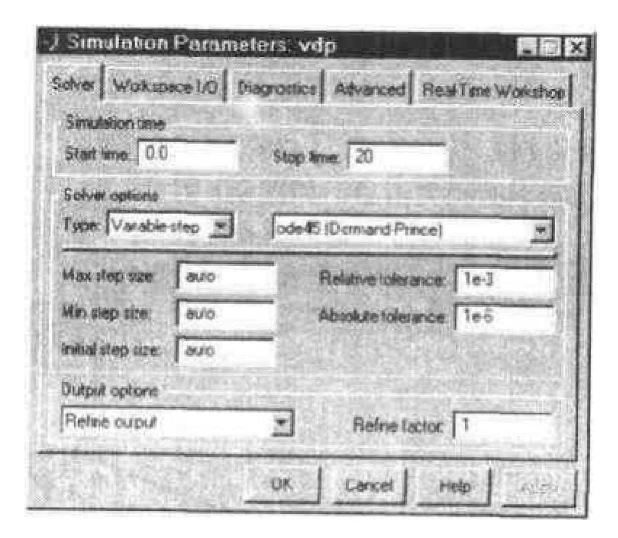


图 9.2　求解器的设置对话框

（1）仿真时间

用户可以修改仿真的开始和结束时间，缺省时仿真从0.0秒开始，在10.0秒处结束。注意这里的仿真时间与实际的时钟时间是不同的，后者是仿真计算所花费的实际时间，它的长短与许多因素有关，如模型的复杂程度、仿真的最大步长以及计算机的时钟频率。

（2）求解器（Solvers）

SIMULINK模型的仿真一般需要采用微分方程或微分方程组的数值解法。SIMULINK为用户提供了各种不同的求解算法，这些算法的使用范围和计算效率各不相同，为了保持仿真的高效性，用户应该根据仿真模型的特点，选择最适合的求解算法。另外，用户还可以在可

变步长或固定步长之间选择。可变步长（Variable-step）提供了误差控制的机制，而固定步长不论每步仿真的误差有多少，总是采用相同的仿真步长。

下面将列举SIMULINK提供的不同求解方法。

① 可变步长类

- ode45 基于4阶或5阶Runge-Kutta 算法，属于一步求解法，即计算当前值y(t n)只需要前一步的结果y(t n−1)，是大多数问题的首选求解算法。is based on an explicit Runge-Kutta (4,5)。
- ode23 基于4阶或5阶Runge-Kutta 算法，属于一步求解法。在较大的容许误差和中度刚性系统模型下比ode45方法更加有效。
- ode113是变阶的Adams-Bashforth-Moulton PECE方法，属于多步预测算法，也就是说计算当前值y(t n)需要前几步的结果。
- ode15s 是基于数值微分公式(numerical differentiation formulas)的变阶算法，属于多步预测算法。如果研究的系统刚度很大，或ode45方法求解的效率不高，可以试试这种算法。
- ode23s 基于改进的二阶公式，属于一步求解法，在较大的容许误差情况下，比ode15s更加有效，因此，如果在求解某些刚性系统时ode15s的计算效果不佳，可以改用这种方法。
- ode23t 一般用来解决中度刚性系统的求解。
- ode23tb 在较大的容许误差情况下可能比ode15s方法有效。
- discrete（variable-step）当系统中没有连续状态变量时选择该方法。

② 固定步长类

- ode5 是采用固定步长的ode45方法。
- ode4 属于4阶Runge-Kutta 方法。
- ode3 采用固定步长的ode23方法。
- ode2 采用改进的Euler公式。
- ode1 即Euler方法。
- discrete（fixed-step）固定步长的离散系统求解算法，尤其适用于不存在状态变量的系统。

（3）输出选项（Output Options）

该选项可以让用户控制仿真产生输出的数目，用户可以在弹出菜单中选择下列选项之一。

- Refine output
- Produce additional output
- Produce specified output only

① Refine output

该选项在仿真结果太差的时候提供更多的输出点，该参数是两个仿真步之间额外输出点的个数。例如，对于调整因子为2的情况，仿真将在相邻两步仿真中间的时刻额外进行一次计算。缺省情况下，该值为1。采用增大调整因子的方法，可以使我们的仿真曲线更加光滑，由于这些额外的输出是通过连续插值完成的，仿真步长并不发生改变，因此仿真速度比采用减小仿真步长的方法快得多。这种方法尤其适合于同ode45方法结合使用。如果我们在绘制系统仿真曲线时由于采用的仿真步长过大使得曲线不光滑，就可以增大调整因子，可以获得更好

的仿真结果。

② Produce additional output

该选项可以让用户直接指定需要增加的额外输出时间，用户可以在相应的时间域中输入具体的时间向量，这些额外增加的输出是通过连续插值实现的。值得注意的是，该选项会根据设置的额外输出调整仿真步长。

③ Produce specified output only

该选项使得SIMULINK只输出指定时刻的仿真结果，它同样会改变仿真步长来适应指定的输出时刻。当我们要在指定的固定时刻比较几个不同的仿真过程时，该选项就很有用了。

9.2.2　工作空间 I/O 的设置

在使用 SIMULINK 进行动态系统仿真时，用户可以直接将仿真结果输出到 MATALB 基本工作空间中，也可以在仿真启动时刻从基本工作空间中载入模型的初始状态，所有这些都是在仿真配置的工作空间属性对话框（如图 9.3 所示）中完成的。

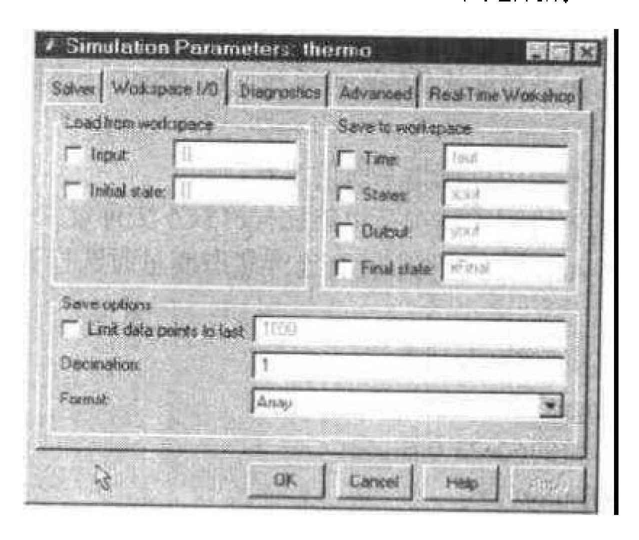


图 9.3　工作空间 I/O 的设置对话框

（1）从基本工作空间中输入数据（Load from Workspaces）

SIMULINK 在运行过程中可以将基本工作空间的数据输入到仿真模型当中，为此，用户需要勾选 “Load from workspace：Input box”，并在相应的 “Input” 栏中输入需要输入的数据变量名，例如[t，u]，该向量的第一列为仿真时间，第二至第 n 列分别对应于模型的第一至第 n 个输入。SIMULINK 在仿真过程中，对于输入的每行数据会根据需要进行插补或跳过来满足仿真步长的需要。

（2）初始状态（Initial state）

勾选该栏，模型将从基本工作空间中获取模型所有内部状态变量的初始值，而不论模块本身是否设置了初始值。该栏中输入的应是基本工作空间中已经存在的变量，变量的次序应与模块中各个状态的次序一致。关于如何获取模型中的状态变量，读者可参考 7.1 节。

（3）保存到工作空间中(Save to workspace)

Time 栏：勾选该栏，模型将把时间变量以指定的变量名输出到基本工作空间中，其缺省名为 tout。

States 栏：勾选该栏，模型将把其所有状态变量以指定的变量名输出到基本工作空间中，其缺省名为 xout。

Output 栏：如果模型窗口中使用输出模块 Out，那么就必须勾选该栏，并填写在基本工作空间中的输出数据变量名。数据的存放方式与输入情况相同。

Final state 栏：勾选该栏，模型将最终状态值输出到基本工作空间，如果最终状态向量在该模型的新一轮仿真中又被用作初值，那么这新一轮仿真是前一轮仿真的继续。

（4）变量存放选项（Save options）

Limit rows to last 栏：该选项可以设定保存变量的数据长度，缺省值为 1000，如果输出数据超过设定值，则最早的历史数据将被“冲掉”。

Decimation 栏：设置解点保存频率。如果取 n，则每隔 n-1 个点保存一个解点。缺省值为 1。

Format 栏：设置保存数据的格式。

9.2.3　诊断页的设置

用户在仿真过程中常常会遇到各种各样的错误或报警消息，其中，错误消息会中止仿真过程，而警告消息则不会。用户可以在诊断属性页中进行适当的设置，来定义是否需要显示相应的错误或报警消息。诊断属性页如图 9.4 所示。

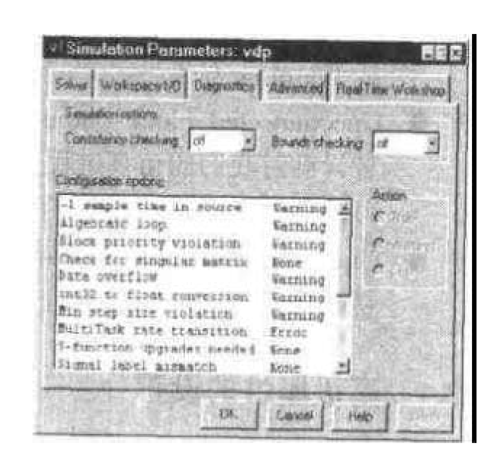


图 9.4　诊断页的设置对话框

（1）一致性检测（Consistency checking）

专门用来调试用户定制模块的编程的正确性，对于由 SIMULINK 标准模块组成的系统一般不需要进行一致性检查，因而一般情况下为 off 状态。

（2）禁止零穿越检测（Disabling Zero Crossing Detection）

勾选该栏，将禁止模块的零穿越检测。所谓零穿越是指 SIMULINK 模块的一种不连续性，例如 Sign 模块在输入为负时，输出为-1，在输入为正时，输出为 1，输出恰好在 0 处发生了突变。这时，如果采用变步长的求解算法进行仿真，当 Sign 模块趋近于 0 时，SIMULINK 将调整积分算法的步长以使 Sign 模块的输出在适当的时候发生改变。这个过程就被称为零穿越检测。零穿越检测提高了仿真的准确性，但影响了仿真的速度。

（3）禁止优化 I/O 存储（Disable optimized I/O storage）

勾选该栏将使 SIMULINK 为每个模块的输入输出分配独立的缓冲区，这样会使系统的内存消耗急剧增加，因此，除了以下的调试情况，一般不勾选该栏。

- 用户在调试 C MEX 形式的 S 函数；
- 在调试时，用户使用浮动窗口观察模型中的信号。

在上述情况下，如果用户勾选此栏，系统将报错。

（4）放松对布尔类型的检查（Relax boolean type checking）

勾选该栏，将使模块本身的双精度输入能够接受布尔类型的输入，这主要是与

SIMULINK 的早期版本（3.0 以前）进行兼容。

随着 SIMULINK 版本的提高，系统将能够自动检测更多的异常情况，对于每种异常，SIMULINK 提供了三种方式进行处理，这可以通过选择对话框右边的选项进行设置。

（1）None 表示 SIMULINK 忽略这种异常。

（2）Warning 表示 SIMULINK 遇到这种异常时发出相应的警告消息，但不中止仿真的进行。

（3）Error 表示 SIMULINK 遇到这种异常时发出相应的错误消息并中止仿真的进行。

下面列举常见的 5 钟异常：

- Algebraic loop：代数环的存在将大大影响模型的仿真速度，对于这种异常，一般采用 Warning 的处理方式，如果用户可以忍受代数环所导致的仿真速度减缓，则可以忽略这种警告，否则可以采取适当的措施消除导数环。
- Min step size violation：这种异常的发生表明微分方程求解器为达到指定的精度要求而需要更小的计算步长，但这又是求解器不允许的。解决的方法是采用更高阶的求解算法。对于这种异常，通常采用 Error 或 Warning 方式。
- Uneconnected block input：该异常的出现表明模型当中存在未被使用的输入端，我们可以将该输入端接到 Ground 模块的输出端来消除这种异常，处理的方式一般为 Error 或 Warning 方式。
- Uneconnected block output：该异常的出现表明模型当中存在未被使用的输出端，这种异常对我们的仿真过程无大的影响，我们只要将该输出端接到 Terminator 模块就可消除这种异常，处理的方式一般为 Error 或 Warning 方式。
- Uneconnected line：这种异常的出现表明模型中存在一端未被使用的信号线，这往往是建模疏忽造成的，处理的方式一般为 Error。

9.2.4　高级属性的设置

高级属性页（如图9.5所示）可以设置仿真配置的某些高级选项，这些选项可以影响 SIMULINK仿真的环境。

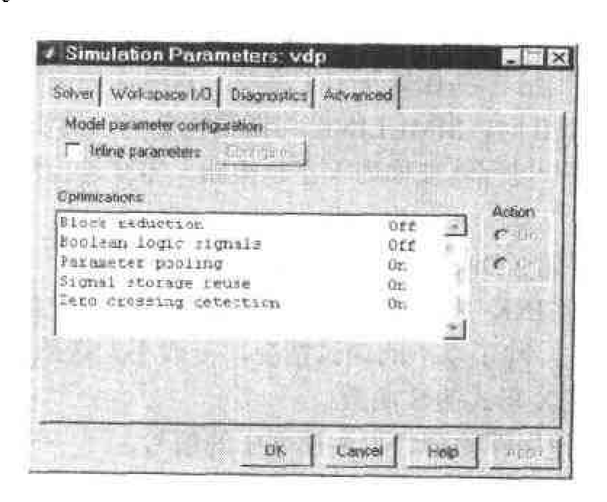


图 9.5　高级属性页的设置

下面具体介绍各项的含义。

（1）Inline parameters（内部参数）

缺省情况下许多模块的参数可以在仿真过程中实时调整。如果勾选该选项，则模型当中所有模块的参数都不能实时调整，用户专门指定的除外。在这种情况下，SIMULINK将这些不能被实时调整的参数视为常数，因此能够提高仿真计算的速度。在该选项被勾选的情况下，要指定某些参数仍旧保持实时调整的性质，可以在模型参数配置（Model Parameter Configuration）对话框中进行设置。弹出该对话框的方法是单击相邻的“Configure”按钮。

如果勾选该项，则只有下面的几种参数能够在仿真过程中实时改变：

- 在 MATLAB 基本工作空间中定义的变量。
- 在模型参数配置（Model Parameter Configuration）对话框中设置为全局（global）变量的参数。

如果要改变以上这些参数值，可以改变相应工作空间中的变量值，然后选择“Edit:Update Diagram”菜单或使用【Ctrl+D】快捷键来更新模型方框图。

如果勾选该项，SIMULINK将常值信号移到仿真循环之外，从而加快仿真的进行。

（2）Optimizations（优化项）

- Block reduction.

尽量用一个综合的模块来替代一组模块集，从而加快模型的执行。

- Boolean logic signals

使得能够接受布尔类型信号的模块仅仅只接受布尔信号。如果该参数为off，则该模块不仅接受布尔信号，同时也能接受双精度型信号。否则，用双精度型数据作为该模块的输入将会出现错误。该选项主要是为了与以前版本的SIMULINK相互兼容。

- Parameter pooling

该选项用于代码生成。一般情况下可以忽略。

- Signal storage reuse

如果该参数为off，则SIMULINK为每一个模块分配独立的内存缓冲区，这样会极大的增加系统内存的消耗，因此一般在进行模型调试时才设置成off状态。另外，以下情况下也一般设置成off状态：

调试一个C MEX文件形式的S函数。

在模型调试过程中使用浮动的示波器（Scope）模块或显示（Display）模块来观察模型中的内部信号值（如果该参数为on时，系统将会报错）。

- Zero-crossing detection

使得SIMULINK在进行变步长积分计算时能够进行零点穿越检测。对于大多数模型，让求解器采取更大的时间步长能够加快仿真的速度。但如果模型的动态变化比较大，则将该参数设置成off虽然能够提高仿真速度，却因此降低了仿真的精度。

（3）Model Parameter Configuration Dialog Box（模型参数配置对话框）

- Source list

显示工作空间中变量的列表，可选项包括：

MATLAB workspace：列出MATLAB工作空间中具有数值量的所有变量。

Referenced workspace variables：只列出该模型中引用的变量。

- Refresh list

如果在上一次进行参数配置之后对工作空间中的变量进行了修改，则单击该按钮可以更

新变量列表的显示。

● Add to table

将source list中选中的变量添加到右边的可调参数列表中。

● New

定义一个新的变量，并且将它添加到可调参数列表中。注意该按钮不会在MATLAB工作空间中创建相应的变量，用户必须另外手动进行创建。

● Storage Class、Storage type qualifier 用于模型的代码生成。

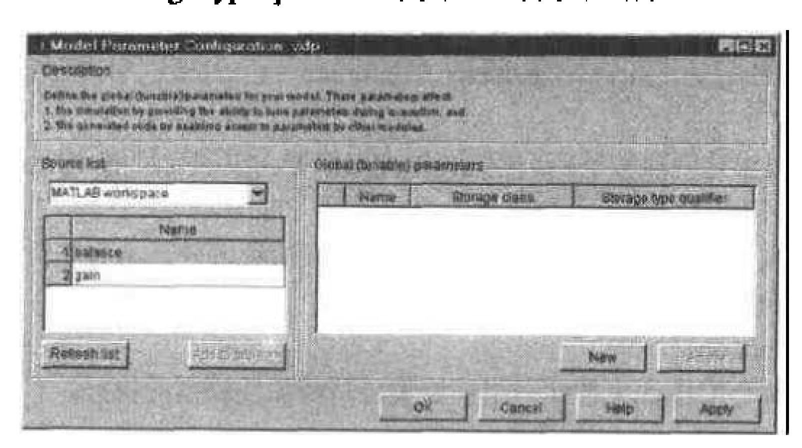


图 9.6　模型参数配置对话框

9.3　优化仿真过程

9.3.1　介绍

SIMULINK 的仿真质量的好坏受到多方面的影响，但归纳起来，一般主要和仿真模型的好坏和仿真参数设置适当与否有关，对于模型的创建，具体问题可以具体分析，这里无法进行统一描述，我们只能从仿真参数的角度来描述影响仿真质量的具体因素。

求解器的缺省设置能够满足大多数问题的速度和精度要求，但是有时用户调整某些求解器或仿真配置的参数，可能会获得更好的仿真结果，尤其是当用户了解系统模型的基本特性，并将这种特性提供给求解器时，有可能很大程度上改善仿真的结果。

9.3.2　提高仿真速度

下列情况将影响实际的仿真速度：

● 模型当中包含MATLAB Fcn模块，这时，由于SIMULINK在每一仿真步中将调用MATLAB解释器，仿真速度会明显降低。解决的办法是尽可能采用内建函数（built-in Fcn）模块或由基本数学模块搭建代替。

● 模型中包含M文件形式的S函数，S函数的执行同样需要调用MATLAB解释器。解决的办法是将S函数转化成一个子系统或C MEX文件形式的S函数。

- 模型当中包含记忆模块，该模块将导致变步长的求解器采用一阶算法来满足仿真时间的要求。
- 最大步长设置太小，导致仿真速度减慢。尽量采用自动（auto）方式。
- 容许误差设置太小，对于大多数情况，其缺省值（0.1%）就足够了，过于追求仿真的精度而将仿真的容许误差设置得太小，会大大增加仿真的时间。
- 仿真时间跨度太大，可改小一些。
- 对于刚性系统采用了非刚性的求解算法，试着采用ode15s算法。
- 模型中的不同采样周期没有倍乘关系，为了满足所有采样周期的需要，SIMULINK可能会采用很小的采样时间，即不同采样周期的最大公约数，这样导致仿真速度的下降。
- 模型中存在代数环。
- 模型中将随机信号连入积分模块中，改进的方法是针对连续系统改用受限白噪声模块。

9.3.3 提高仿真精度

为了检测仿真的准确性，我们可以将仿真时间设置在一段合理范围运行仿真，然后将相对容许误差减小到1e-4（缺省值为1e-3），再次运行仿真，比较前后两次仿真的结果，如果最终的仿真结果没有什么不同，我们就可以初步确定仿真结果是收敛的。

如果仿真没有反映系统启动时刻的动态特性，可以减小仿真的初始仿真步长，使得仿真不至于跳过系统的某些关键特性。

如果得到的仿真结果不稳定，则可能有以下情况：

- 系统本身是不稳定的。
- 如果使用ode15s的求解方法，确保最大阶次为2，或者改用ode23s算法。

如果仿真结果看起来还没有达到所需要的精度，则可以根据情况采取下列措施：

- 对于存在零附近状态的系统，如果设置的绝对容许误差太大，仿真过程可能会在零附近的区域反复进行迭代。解决的方法是减小该值，或在相应的积分模块中单独进行设置。
- 如果减小绝对容许误差不足以提高仿真的精度，可调小相对容许误差的大小来强制减小仿真的步长从而增加仿真的步数。

9.4 从命令窗口中执行仿真

9.4.1 介绍

除了在模型窗口中使用菜单命令启动仿真外，MATLAB 还允许在命令窗口中输入指令或编写 M 文件来实现模型的仿真，这种方法相比菜单方式具有更好的灵活性，例如可以实时改变模型的参数，研究不同参数、输入和初始条件的影响等。

9.4.2　使用 sim 指令

在命令窗口中运行仿真的指令是 sim，具体使用格式是：

[t，x，y] = sim('mode')

[t，x，y] = sim('model'，timespan，options，ut)

[t，x，y1，y2，…，yn] = sim(' model'，timespan，options，ut)

各参数说明如下：

model：进行仿真的模型名，不包含扩展名。该模型文件必须位于 MATLAB 的搜索路径上。

y：输出矩阵，对应于模型中输出口模块的记录。y 的第 k 列对应于模型中第 k 个输出口的数据。y1，y2，…，yn 都是列向量，分别对应于 n 个输出口的数据。

X：状态矩阵，每一列表示相应状态的记录，

timespan：指定仿真的时间区间。可以采用下面的几种格式：[]表示使用模型对话框中的设置时间。T_final 指定仿真的结束时间；[T_start　T_final]采用向量形式分别指定仿真的开始和结束时间。

options：MATLAB 的特定数据结构，用于指定具体的仿真参数，可以覆盖模型对话框中的设置。采用 simset 和 simget 指令可以对它进行设置和读取。

还可以用 set_param 指令来启动、暂停或继续仿真过程，也可以用它来更新某个模块，同时还可以用它来获取仿真的具体状态。具体用法是：

set_param('sys', 'SimulationCommand', 'cmd')

其中，'sys'是系统名称，　'cmd'则是'start', 'stop', 'pause','continue',或'update'命令之一。

get_param 指令也具有类似的用法：

get_param('sys', 'SimulationStatus')

SIMULINK将返回 'stopped', 'initializing', 'running', 'paused','terminating'和 'external'（与 Real-Time Workshop 一同使用）之一。

例9.1 采用指令方式进行Van der Pol方程的仿真

```
>>[t,x,y] = sim('vdp', 1000, simset('MaxRows', ...
   100,'OutputVariables', 'ty', 'FinalStateName',  'xFinal'));
>>plot(x)
```

9.4.3　仿真配置指令的使用

simget 和 simset 是两个与 sim 配用的指令，前者可以获取仿真模型的属性，后者则可以对模型设置新的选项。

simget 的使用格式为：

```
struct = simget(model)                  %获取模型当前的仿真参数设置
value = simget(model, property)         %获取模型中指定参数的值
```

simset 的使用格式为

```
options = simset(property, value, ...);     %将参数property设置成新的值value，修改
```

```
                                              后的结果保存在options中
options = simset(old_opstruct, property, value, ...); %将old_opstruct的参数property设置成新
                                              的值value，修改后的结果保存在options
                                              中
options = simset(old_opstruct, new_opstruct);   %结合old_opstruct和new_opstruct中的参
                                              数，两者重复时以后者的值为准。
simset                                        %显示所有的可设置的参数名、取值选项
                                              和缺省值
```

例9.2 设置和获取Van der Pol的属性

（1）设置属性

```
>>myopts = simset('MaxRows', 100, 'Refine', 2);
>>[t,x,y] = sim('vdp', 10, myopts);
>>plot(y)
```

其结果如图 9.7 所示。

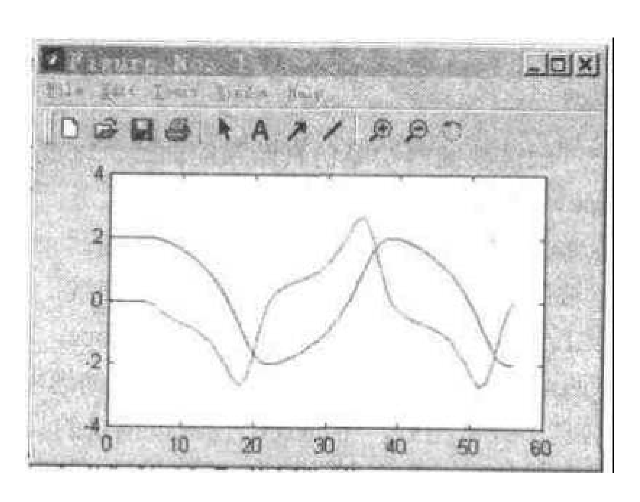


图 9.7　Van der Pol 方程的仿真曲线

（2）获取属性

```
>>options = simget('vdp')
options =
            AbsTol: 1.0000e-006
             Debug: 'off'
        Decimation: 1
      DstWorkspace: 'current'
    FinalStateName: ''
         FixedStep: 'auto'
      InitialState: []
       InitialStep: 'auto'
          MaxOrder: 5
        SaveFormat: 'Matrix'
```

```
          MaxRows: 0
          MaxStep: 'auto'
     OutputPoints: 'all'
  OutputVariables: ''
           Refine: 1
           RelTol: 0.0010
           Solver: 'ode45'
     SrcWorkspace: 'base'
            Trace: ''
        ZeroCross: 'on'
```

9.5 仿真结果的观察

9.5.1 使用示波器

仿真进行当中，用户一般需要随时绘制仿真结果的曲线，以观察信号的实时变化，在模型当中使用示波器（Scope）是其中最为简单和常用的方式。

Scope 模块可以在仿真进行的同时用来显示输出信号曲线。

在 Scope 模块中进行输出显示还不完善，它只显示输出曲线，而没有任何标记。其他的一些 Graph Scope 模块，如 Auto-Scale Graph Scope，Graph Scope，XY Graph Scope 模块除了显示输出曲线外，还显示两个轴和彩色的线型，不过它们执行的速度要比 Scope 模块慢得多。

由于示波器模块在仿真中经常用到，下面我们说说该模块的具体使用方法。

示波器模块可以接受向量信号，在仿真过程中，实时显示信号波形，如果是向量信号，它还可以自动以多种颜色的曲线分别显示向量信号的各个分量。

不论示波器是否已经打开，只要仿真一启动，示波器缓冲区就会接受传递来的信号。该缓冲区数据长度的缺省值 5000。如果数据长度超过设定值，则最早的历史数据将被冲掉。

在示波器窗口（如图 9.8）中，三个图标分别表示 X-Y 双轴调节、X 轴调节和 Y 轴调节。图标可以根据数据的实际范围自动设置纵坐标的显示范围和刻度。双击图标则打开示波器属性对话框。

在示波器的显示范围内，单击鼠标右键，将弹出一个上下文菜单，选择“Axes properties”菜单项，将弹出纵坐标设置对话框。在相应的输入栏中输入所希望的纵坐标上下限，可以调整示波器实际纵坐标显示的范围。

示波器属性对话框如图 9.9 所示，其各部分的设置为：

- Time range：缺省值为 10，表示显示数据的区间在[0，10]内，如果数据的实际范围超出设定的区间，则超出的部分不再显示。
- Sampling：包含两个下拉菜单。Decimation 表示显示频度。如果取 n，则每隔（n-1）个数据点给与显示。缺省值为 1。Sample time 表示点的采样时间步长。缺省值为 0，表示显示连续信号。如果取-1，则表示显示方式取决于实际输入信号。如果取大于零

的整数，则表示显示离散信号的时间间隔。

- Limit rows to last：设定缓冲区接受数据的长度。缺省为选中状态，其值为 5 000。
- Save data to workspace：将示波器缓冲区中保存的数据送入 MATLAB 基本工作空间中。保存的变量名可以修改。
- Number of axes：缺省值为 1，此时 Scope 模块只有一个输入口，示波器窗口只有 1 个信号显示区。如果值为 2，Scope 模块有 2 个输入口，示波器窗口有 2 个信号显示区，以此类推。
- Floating scope：示波器是否以浮动窗口形式出现。

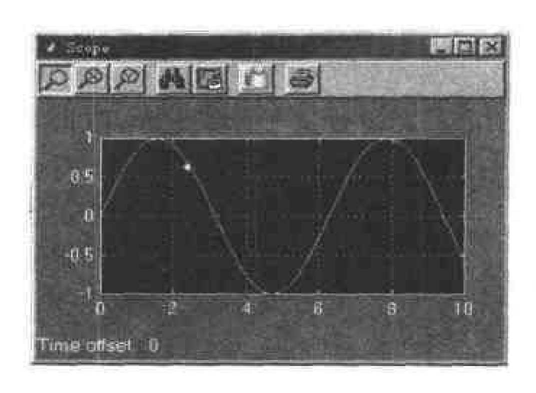


图 9.8　Scope 模块的显示窗口

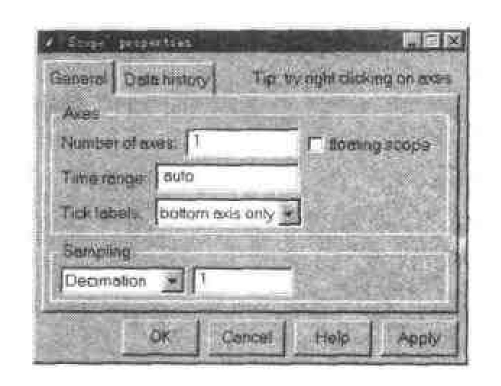


图 9.9　Scope 模块的属性对话框

9.5.2　使用返回变量方式

通过返回仿真时间和输出历史数据，还可以使用画图指令来显示仿真结果，如图 9.10 所示的模型。

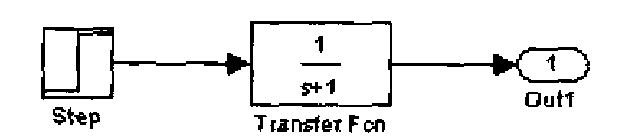


图 9.10　使用返回变量形式来分析仿真结果

名字为 y 的模块是 Signals & Systems 库中的标准模块，输出轨迹 y 由积分函数返回。把参数 Return Variable 定义为[t,x,y]，然后就可以使用下面的命令来画图：

```
>>plot(t,y)
```

9.5.3　使用工作空间方式

To Workspace 模块的功能是把输出曲线送到 MATLAB 的基本工作空间里的变量中。图 9.11 演示的就是 To Workspace 模块的功能。

当仿真结束后，变量 y 和 t 就存放在工作空间中。时间变量是通过把 Clock 模块的输出送到 To Workspace 模块来保存的。

To Workspace 模块还可接受一个向量的输入，每个输入元素的轨迹是以一个列向量的形

式存放在工作空间的变量中。

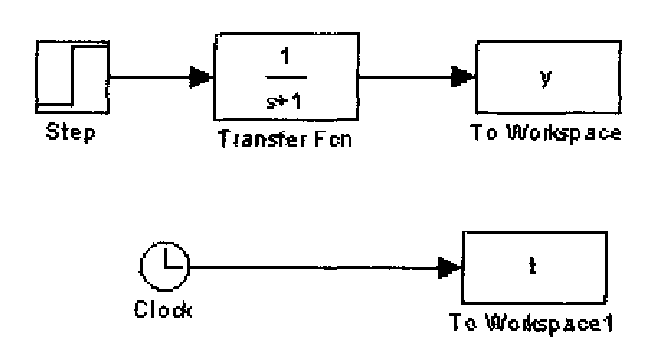


图 9.11　使用工作空间方式分析仿真结果

第 10 章　模型的调试

用户创建的仿真模型常常存在这样或那样的错误，即所谓的 Bug，其中有的错误是由于用户在创建模型过程中的疏忽造成的，这些错误在仿真启动前就能很容易被系统检测出来，但是还有一些错误是因为模型本身的不合理或与原系统存在出入，最终导致仿真结果不正确，甚至影响仿真过程的正常进行，这时只有通过对仿真模型的一步步的调试，才有可能发现它们并进行修正。

SIMULINK 4.0 提供了强大的模型调试功能，并且在 3.0 版本的基础上增加了图形界面的支持，使得用户对模型的调试和跟踪更加方便。

本章主要讲述如何利用 SIMULINK 自身的调试功能对创建的模型进行调试，包括指令方式和图形界面方式。目的是使读者熟悉 SIMULINK 4.0 的调试环境，并初步掌握模型调试的基本操作和方法。

10.1　SIMULINK 4.0 的调试环境

10.1.1　启动调试器

有两种不同的方法启动调试器：

菜单方式：在模型窗口中选择"Tool：Debugger"菜单项，将出现如图 10.1 所示的调试窗口。



图 10.1　SIMULINK 的调试窗口

指令方式：相关的指令为 sim 和 sldebug，如：

```
>>sim('vdp',[0,10],simset('debug','on'))
```

>>sldebug 'vdp'

上述指令将装载 SIMULINK 的演示模型之一：vdp.mdl，启动仿真过程，并在第一个模块处停止，同时打开调试器。

10.1.2　开始调试

为了启动仿真，可以选择调试器窗口的“Start/Continue”按钮，这时仿真过程开始，并在执行的第一个模块处暂停，如果这时模型窗口没有打开，SIMULINK 将自动打开模型文件，并在仿真暂停处高亮显示相关的模块及信号线（此处为第一个模块），如图 10.2 所示。如果用户采用指令方式启动调试过程，调试器将自动在命令窗口中输出仿真开始的时间及命令提示。如果是采用菜单方式，这些信息将在调试器的 Output 输出窗口中显示。

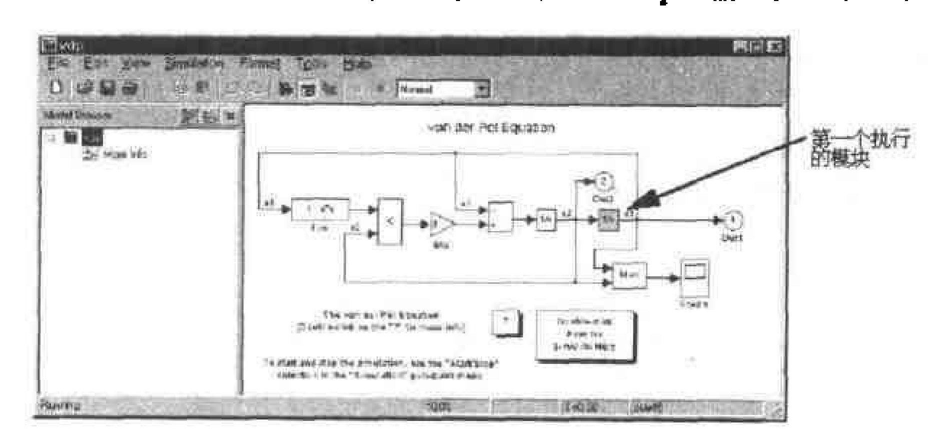


图 10.2　仿真在第一个模块前挂起

> 注意：如果用户采用菜单方式激活调试过程，该模型在命令窗口中的调试过程也会同时自动地激活，但为了保持调试过程的同步，这时应该尽量避免使用指令方式。

启动调试过程后，用户可以设置断点、单步执行仿真、直接运行到下一个断点、检查工作空间中的变量或者其他的调试工作。

如果采用指令方式，用户可以在命令窗口中输入适当的调试命令来控制调试过程的进行。

SIMULINK 为了方便用户表示模型中的各个模块，在许多调试指令当中可以采用模块索引来代替模块本身，模块索引具有 s:b 的结构，其中，s 表示子系统名，对于根系统该值为 0；b 表示子系统当中的第几个模块。例如：0:1 表示 0 系统的索引号为 1 的模块。使用 slist 指令可以显示正在调试的模型中所有的模块索引。

命令窗口在用户进入调试阶段后，将出现 sldebug 的提示信息，随后将显示目前模型中将要执行的模块名等信息。在调试过程当中，用户还可以在 sldebug 的提示下输入任何的 MATLAB 指令。例如，假设仿真运行到某个断点处，我们需要绘制 tout、yout 的相关曲线，可以在命令窗口中输入：

>>(sldebug ...) plot(tout, yout)

如果我们需要观察某个变量的当前值，例如 v，可以输入：

>>(sldebug ...) v

这里有一点需要注意，如果我们输入的变量名与某个指令的完整或缩写形式相同，则调

试器忽略变量的现实，而执行相应的指令，如：

```
>>(sldebug ...) s
```

表示单步执行仿真，而不是显示 s 变量的值。如果一定要显示 s 变量，可以输入：

```
>>(sldebug...) eval(s')
```

10.1.3 获取在线帮助

如果读者对 SIMULINK 的调试过程不是很熟悉，可以通过调试器的在线帮助获得相关的使用介绍。方法是单击调试器窗口中的在线帮助按钮。当然，用户还可以随时按下【F1】键来直接获得上下文的帮助信息。

10.2 调试过程

10.2.1 调试步骤

一旦仿真在某个断点处挂起后，调试器允许用户继续运行仿真，直到：

- 仿真结束；
- 下一个断点处；
- 下一个模块；
- 下一步仿真。

如果采用菜单方式，用户只要在调试器窗口中单击相应的图标。

当然，用户也可以在命令窗口中输入相应的指令来实现。

以下是相关的指令：

step	执行道下一个模块处。
next	执行道下一步仿真处。
continue	执行到下一个断点处，如果后面没有断点则执行到仿真结束。
run	无论后面有没有断点，强制执行到仿真结束。

1. 执行到下一个模块处

单击图标，或者在命令窗口中输入 step 指令，仿真将向下执行到相对当前模块的下一个模块处。这时模型将下一个模块高亮显示，表示这是下一个将要执行的模块。SIMULINK 每执行一个模块，都会在调试器的 output 窗口中显示该模块的输入（U）和输出（Y），随后显示的 sldebug 提示了下一个将要执行的模块，（如果是指令方式，则在命令窗口中显示）。例如，下面是对 vdp 模型调试的结果：

```
>>(sldebug @0:0 'vdp/Integrator1'): step
U1 = [0]
Y1 = [2]
(sldebug @0:1 'vdp/Out1'):
```

当 SIMULINK 执行完最后一个模块时，调试器将仿真过程挂起，等待下一步仿真的执行。

这时，为了通知用户已经进入以每步仿真单位（不再以模块为单位）的调试过程，调试器将在调试器的 output 窗口（图形方式）或命令窗口（指令方式）输出当前的时间。例如本例中，当用户单步执行完最后的模块时，output 窗口将显示：

```
>>(sldebug @0:8 'vdp/Sum'): step
U1 = [2]
U2 = [0]
Y1 = [-2]
[Tm=0.0001004754572603832 ] **Start** of system 'vdp' outputs
```

用户还可以在模块当中以最小仿真时间为单位进行单步调试，这时用户必须在调试器左下方的 Break on conditions 栏中勾选 Minor time steps 项或在命令窗口中输入 minor 指令。

2. 执行到下一步仿真

为了执行到下一步仿真，单击图标或在命令窗口中输入 next 指令。调试器将执行当前的仿真步，然后在下一步仿真开始前挂起。例如，本例开始进行 vdp 模型的调试后，输入 next 指令，将显示目前的仿真时间：

```
[Tm=0.0001004754572603832 ] **Start** of system 'vdp' outputs
```

10.2.2　无条件断点的设置

调试器允许用户在模型中设置断点，并且借助 continue 指令可以将仿真从一个断点直接执行到下一个断点。SIMULINK 模型中的断点分为两类：无条件断点和条件断点。如果用户在指定模块或仿真时间处设定无条件断点，则每当仿真执行到用户指定的模块或仿真时间处都会挂起。而条件断点则只是在某种条件得到满足时才起作用。

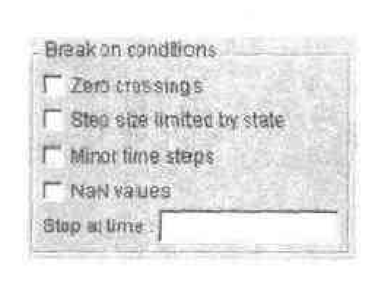


图 10.3　条件断点的设置

如果用户感觉模型的某处或某种条件下的执行有问题，可以在该处手工设置断点，启动调试过程后，通过 continue 指令，仿真将会直接运行到该处，用户就可以通过单步执行来观察仿真具体的计算过程了。

设置断点的方法也有图形和指令两种方式，在图形方式下，如果是无条件断点，用户可以选择工具条上的图标，如果是条件断点，在可以在调试器的左下栏中勾选合适的断点条件，如图 10.3 所示。

如果采用指令方式，则可以输入以下的指令：

break <gcb | s:b>	在仿真开始设置断点；
bafter <gcb | s:b>	在仿真末尾设置断点；
tbreak [t]	在仿真的某个时刻（仿真步）；
nanbreak	在发生溢出时，即出现NaN或Inf值时；
xbreak	当某个模型状态限制了仿真步长时；
zcbreak	当相邻两步仿真之间发生零点穿越现象时。

调试器还允许用户在模块的执行开始（图形或指令方式均可）或结束时（指令方式）设置断点。

如果想在当前选中模块开始之前设置断点，可以输入指令

>>break gcb

也可以通过模块索引来指定其他模块:

>>break s:b

注意：我们不能在虚拟模块开始或结束时设置断点，因为虚拟模块只有图形上的意义，可以使用 slist 指令来了解模型当中哪些模块不是虚拟模块。

在调试器的 Watch points 栏可以显示设置了断点的模块名。

通过指令 bafter，用户还可以在模块执行结束时设置断点。

10.2.3　无条件断点的清除

在图形方式下，如果只是暂时清除某个模块的断点，只要在 Watch points 栏中将相应的模块置于不选中状态。如果是永久清除断点，则可以单击“Remove watch”按钮来实现。

在指令方式下则可以通过 clear 指令来清除断点：

>>clear s:b

>>clear gcb

10.2.4　条件断点的设置

1. 在某个仿真时刻设置断点

在图形方式下，只要在调试器中的 Stop at time 栏输入需要设置断点的仿真时刻。在指令方式下，则可以采用 tbreak 指令。

例如，在本例中，我们在命令窗口输入：

>> tbreak 9

>>continue

[Tm=9.07847133212036] **Start** of system 'vdp' outputs

这表示首先在 9 秒时刻设置了断点，然后继续执行（以仿真步长为单位）到下一个时刻 9.0785 处挂起。

2. 在系统出现溢出时设置断点

在调试器的 Break on conditions 栏中勾选 NaN values，或在命令窗口中输入 nanbreak 指令，将使系统发生溢出时中断仿真过程。

3. 在模型状态限制了仿真步长时设置断点

在调试器的 Break on conditions 栏中勾选 Step size limited by state，或在命令窗口中输入 xbreak 指令，则当使用变步长算法，而模型状态限制了仿真步长时，仿真挂起。

4. 在检测到零穿越时设置断点

在调试器的 Break on conditions 栏中勾选 Zero crossings，或在命令窗口中输入 zcbreak 指令，则当仿真进行过程当中检测到非采样零穿越现象发生时，仿真挂起。仿真过程挂起后，可以输出零穿越发生的位置、时间、类型（上升还是下降）等信息。

例如，我们在调试 zeroxing 延时模型时输入：

>>sldebug zeroxing

[Tm=0] **Start** of system 'zeroxing' outputs
(sldebug @0:0 'zeroxing/Sine Wave'): zcbreak
>>(sldebug @0:0 'zeroxing/Sine Wave'): continue
[Tm=0.34350110879329] Rising zero crossing on 3rd zcsignal
in block 0:2 'zeroxing/Saturation'

如果模型当中没有可能出现这种现象的模块，SIMULINK 也会通知用户。

10.3 仿真信息的显示

10.3.1 模块 I/O 的显示

SIMULINK 允许在调试过程中显示某个模块的输入、输出或状态变量。方法是单击相应的图标。或在命令窗口中输入下面的指令：

probe	立刻输出模块的信息
disp	在每个断点处输出
trace	每当模块执行时输出

1. 输出指定模块的 I/O

选中相应的模块，然后单击图标，调试器将输出选中模块的 I/O 信息。如果采用指令方式，可以使用 probe 指令，其用法有三种：

>> probe

% 进入或退出 probe 模式。在该模式下，调试器输出任何用户在模型中选中的模块的当前输入和当前输出。输入任何其他命令也可使调试器退出 probe 模式。

>> probe gcb

%显示当前选中模块的输入输出。

>>probe s:b

%显示由 s：b 索引指定模块的输入输出。

从上面的介绍我们知道，当我们以模块为单位一步步执行模型的仿真过程时，调试器每执行完一个模块都会自动显示该模块的输入和输出，借助 probe 指令我们还可以了解其他模块的输入输出。同时当我们以仿真的步长为单位单步执行仿真时，模块的输入输出就不会自动显示。这时借助 probe 同样可以获取模块的输入和输出。

2. disp 指令

当仿真挂起时，使用disp指令可以使调试器显示指定模块的输入和输出。指定模块的方法可以采用模块索引，也可以直接在框图中选择，然后用gcb得到当前选中的模块。用户还可以使用undisp指令来清除调试器显示列表中的模块。例如输入

>>block 0:0

disp指令还可以在单步执行仿真时监视某个或某几个模块的I/O。使用该指令，用户可以指定需要监视的模块，调试器在执行的每一步都会自动显示该模块的输入输出。当然，正如前面所说的，如果用户以模块为单位调试程序，则当前模块的输入输出会在每一步仿真后自

动显示，如果用户仅仅需要的是当前模块的I/O，就没有必要使用disp指令了。

3. trace 指令

trace 指令可以使调试器每当执行到某个模块时就显示该模块的输入输出。这使得用户可以在不中止仿真过程的情况下获得指定模块的完成的输入输出信息。借助其他的指令，用户还可以浏览其他模块的输入输出。在调试器跟踪断点列表中清除指定模块的方法是使用 untrace 指令。

10.3.2　代数环的显示

借助atrace指令，用户可使调试器每当执行到代数环时，显示该代数环的相关信息。附加的参数可以控制显示的信息内容。如：

atrace 0	不显示任何信息
atrace 1	显示计算代数环所需要的迭代次数，估计的计算误差。
atrace 2	与1相同
atrace 3	在2的基础上显示计算代数环所使用的矩阵
atrace 4	在 3 的基础上显示代数环变量的当前解。

10.3.3　显示系统状态

states指令可以在MATLAB命令窗口中显示系统所有状态的当前值。例如，对于SIMULINK中的演示程序bounce.mdl，下列的指令可以在第一步和第二步仿真后显示状态的实际值。

```
>>sldebug bounce
[Tm=0 ] **Start** of system 'bounce' outputs
>>(sldebug @0:0 'bounce/Position'): states
Continuous state vector (value,index,name):
10 0 (0:0 'bounce/Position')
15 1 (0:5 'bounce/Velocity')
>>(sldebug @0:0 'bounce/Position'): next
[Tm=0.01 ] **Start** of system 'bounce' outputs
>>(sldebug @0:0 'bounce/Position'): states
Continuous state vector (value,index,name):
10.1495095 0 (0:0 'bounce/Position')
14.9019 1 (0:5 'bounce/Velocity')
```

10.4　模型信息的显示

10.4.1　显示模型中模块的执行次序

使用slist指令，用户可以在调试过程中随时按执行次序显示模型中的所有执行模块，包括它们的索引号。例如，在vdp模型的调试过程中，输入slist指令，可以得到下面的信息：

```
---- Sorted list for 'vdp' [12 blocks, 9 nonvirtual blocks,
directFeed=0]
0:0 'vdp/Integrator1' (Integrator)
0:1 'vdp/Out1' (Outport)
0:2 'vdp/Integrator2' (Integrator)
0:3 'vdp/Out2' (Outport)
0:4 'vdp/Fcn' (Fcn)
0:5 'vdp/Product' (Product)
0:6 'vdp/Mu' (Gain)
0:7 'vdp/Scope' (Scope)
0:8 'vdp/Sum' (Sum)
```

10.4.2　根据索引号确定模块

如果用户不知道某个索引号对应哪一个具体模块，可以输入 bshow s:b 指令，结果调试器将根据索引号寻找所对应的模块，如果该模块放置在一个没有打开的子系统中，调试器将自动打开并显示该子系统，同时选中所要寻找的模块。

10.4.3　显示模型当中的非虚拟系统

systems指令可以用来显示模型当中的非虚拟子系统，如对于clutch的演示模型：

```
>>sldebug clutch

[Tm=0 ] **Start** of system 'clutch' outputs
>>(sldebug @0:0 'clutch/Clutch Pedal'): systems
0 'clutch'
1 'clutch/Locked'
```

注意：systems 指令不会显示模型中的一般子系统(Subsystem)。这是因为一般子系统没有对子系统中的内容进行抽象，而只是为了表达方便在图形中加以区分。一般子系统中的模块仍然属于上一层系统中的内容，因此不会在该指令中显示。一般而言，除了根系统(root system)、条件子系统以外，其他的子系统都属于虚拟系统，因而不在 systems 指令中显示。

10.4.4　显示模型当中的非虚拟模块

在 SIMULINK 模型中，诸如 Mux、Demux 等模块只是用来处理信号之间的分解、合并或者分流，它们本身并不完成某个具体的系统功能，SIMULINK 只是在把它们作为虚拟的系统，我们称之为虚拟模块。虚拟模块在调试中不会单独占用仿真时间，因此我们有必要知道模型当中究竟那些模型不是虚拟模块或系统，systems 指令可以帮助我们解决这个问题，它能够列出模型中所有非虚拟系统的名称。例如在 Simlunk 的演示例子 vdp 当中输入：

```
>>sldebug vdp
[Tm=0 ] **Start** of system 'vdp' outputs
>>(sldebug @0:0 'vdp/Integrator1'): slist
---- Sorted list for 'vdp' [12 blocks, 9 nonvirtual blocks,
```

```
directFeed=0]
0:0 'vdp/Integrator1' (Integrator)
0:1 'vdp/Out1' (Outport)
0:2 'vdp/Integrator2' (Integrator)
0:3 'vdp/Out2' (Outport)
0:4 'vdp/Fcn' (Fcn)
0:5 'vdp/Product' (Product)
0:6 'vdp/Mu' (Gain)
0:7 'vdp/Scope' (Scope)
0:8 'vdp/Sum' (Sum)
```

10.4.5 显示零点穿越模块

借助 zclist 指令，还可以将模型当中所有可能发生零点穿越现象的模块列举出来。同样对于 clutch 模型：

```
>>(sldebug @0:0 'clutch/Clutch Pedal'): zclist
2:3 'clutch/Unlocked/Sign' (Signum)
0:4 'clutch/Lockup Detection/Velocities Match' (HitCross)
0:10 'clutch/Lockup Detection/Required Friction
for Lockup/Abs' (Abs)
0:11 'clutch/Lockup Detection/Required Friction for
Lockup/ Relational Operator' (RelationalOperator)
0:18 'clutch/Break Apart Detection/Abs' (Abs)
0:20 'clutch/Break Apart Detection/Relational Operator'
(RelationalOperator)
0:24 'clutch/Unlocked' (SubSystem)
0:27 'clutch/Locked' (SubSystem)
```

10.4.6 显示代数环

ashow 指令可以将模型中指定代数环或包含指定模块的代数环用高亮形式显示出来。如果要高亮显示指定的代数环，可以输入 ashow s#n，其中，s 是子系统的索引，n 是该子系统中现有相应环的索引。如果要高亮显示包含当前选中模块的代数环，可以输入 ashow gcb 指令；当然我们还可以使用模块的索引号来高亮显示包含索引号所对应的模块的代数环，即 ashow s:b。输入 ashow clear 指令可消除代数环的高亮显示。

10.4.7 显示调试器设置信息

status指令可用于显示各种调试选项的当前设置，包括条件断点。我们同样考察vdp的模型。

```
>>sim('vdp',[0,10],simset('debug','on'))
[Tm=0 ] **Start** of system 'vdp' outputs
>>(sldebug @0:0 'vdp/Integrator1'): status
Current simulation time: 0 (MajorTimeStep)
Last command: ""
Stop in minor times steps is disabled.
Break at zero crossing events is disabled.
```

Break when step size is limiting by a state is disabled.
Break on non-finite (NaN,Inf) values is disabled.
Display of integration information is disabled.
Algebraic loop tracing level is at 0.

10.4.8　调试命令列表

由于 MATLAB 旧版本的用户一般习惯于采用指令方式进行模型的调试，为了方便大家的使用，这里将调试过程中有可能用的指令一一列出，限于篇幅，只给出简要的功能说明，读者如果需要了解详细的使用方法，可以参考 Sl_using.PDF 文档或者借助 help 指令。

ashow（as）	显示代数环；（括号中的为指令的缩写形式，以下同）
atrace（at）	设置代数环的跟踪水平
bafter（ba）	在某个模块的执行后插入断点
break（b）	在某个模块的执行前插入断点
bshow（bs）	显示指定的模块
clear（cl）	清除某个模块的断点
continue（c）	继续仿真过程
disp（d）	当仿真挂起时显示模块的I/O
help（? 或 h）	显示调试命令的帮助信息
ishow（i）	允许或禁止积分信息的显示
minor（m）	设置或退出最小仿真步模式
nanbreak（na）	设置或清除溢出的条件断点
next（n）	执行到下一个仿真步的开始
probe（p）	显示某个模块的I/O
quit（q）	退出仿真
run（r）	运行仿真直到仿真结束
slist（sli）	列举模型中的非虚拟系统
states（state）	显示当前的状态值
status（stat）	显示当前设置的调试器选项
step（s）	单步执行到下一个模块
stop（sto）	停止仿真
systems（sys）	列举模型中的非虚拟系统
tbreak（tb）	设置或清除时间断点
trace（tr）	每当指定模块执行时显示该模块的I/O
undisp（und）	从调试器的显示列表中清除某个模块
untrace（unt）	从调试器的跟踪列表中清除某个模块
xbreak（x）	设置的step-size-limiting state条件断点
zcbreak（zcb）	设置的zero-crossing条件断点
zclist（zcl）	列举所有可能发生零点穿越的模块

第 11 章　S 函数的编写

MATLAB中的S函数（system-functions）为用户提供了扩展SIMULINK功能的一种强大的机制。通过编写S函数，用户可以向SIMULINK模块中添加自己的算法，该算法可以采用MATLAB语言编写，也可以采用C语言实现。只要遵循一定的规则，我们就可以在S函数中实现任意的算法。在完成S函数的编写后，将函数名放在SIMULINK的S-Function模块中，并且通过封装可以为该模块定制合适的“外衣”。实际上，S函数不仅可以实现几乎所有的SIMULINK标准模块功能，还可以在SIMULINK中完成诸如动画显示等特殊功能的模块。

S函数的功能尽管强大，但由于S函数的编写涉及到SIMULINK仿真过程的部分具体细节，要求用户对SIMULINK的运行机制有深入的了解，因此S函数的使用属于SIMULINK较高的应用层次。一般情况下可以用基本模块实现的功能尽量不使用S函数，但在实际情况中可能会遇到需要编写S函数的情况，因此本章将简要介绍如何进行S函数的编写，所采用的例子都是相对简单的，旨在是用户能初步掌握S函数的编写方法，如果读者需要进一步掌握S函数的创作方法，就需要在实践中多加练习和比较。

11.1　S 函数概述

11.1.1　什么是 S 函数

S函数是一种采用MATLAB或C语言编写，用以描述动态系统行为的算法语言。采用C语言编写的S函数还可以通过mex指令编译成MEX文件，该文件可以在需要时自动连接到MATLAB中去。

S 函数采用一种特殊的调用规则，从而让用户能够同 SIMULINK 自身的方程求解器进行交互，这种交互过程同 SIMULINK 本身标准模块的工作机制几乎完全相同。S 函数支持连续系统、离散系统以及混合系统，因此几乎所有的 SIMULINK 模块都可以采用 S 函数实现。

11.1.2　什么时候使用 S 函数

S 函数最常用的功能是创建定制的 SIMULINK 模块，用户可以采用 S 函数实现：

- 加入新的通用模块到 SIMULINK 当中。
- 将已存在的 C 代码形式的算法加入到 SIMULINK 当中。
- 采用方程组的形式来描述动态系统。
- 实现图形的动画显示。

使用 S 函数最大的优点在于用户可以创建通用的 SIMULINK 模块，该模块可以在同一个

模型当中被多次使用，而不同的模块可以根据情况设置不同的参数。

11.1.3 S函数的工作原理

SIMULINK当中的任何模块都是由输入矢量u、输出矢量y和状态矢量x三部分构成。各个矢量的状态可以是连续的或离散的，也可以是连续离散混合的信号。输入、输出和状态之间的数学关系可以表示成：

$$
\begin{aligned}
y &= f_0(t,x,u) \\
\dot{x}_c &= f_d(t,x,u) \\
x_{d_{k+1}} &= f_u(t,x,u)
\end{aligned}
\tag{11.1}
$$

其中，
$$x = x_c + x_d$$

在以M文件形式编写的S函数中，状态变量被分成两个部分，占据第一部分的是连续状态变量，而后一部分则是离散状态变量。如果该模块没有状态变量，则x为空。在以MEX文件形式编写的S函数中，将存在两个相互独立的状态矢量，分别对应于连续的状态和离散的状态。

SIMULINK在仿真过程中每隔一段时间就会对模型中的所有模块进行调用，每个模块在调用过程中都会完成诸如输出信号的计算、内部状态变量的更新及其导数的计算等工作。而在仿真开始和结束时刻的调用将完成该模块的初始化和扫尾工作。图显示了SIMULINK是如何完成仿真过程的。首先，SIMULINK对所有的模块进行初始化，包括S函数。然后，SIMULINK进入仿真循环，每一次循环被称之为一步仿真，在每一步仿真中，SIMULINK将执行其中的S函数，直到仿真结束。

SIMULINK在仿真过程中反复调用S函数，在调用过程中，SIMULINK将调用S函数子程序，这些子程序将完成以下工作：

- 在仿真循环开始之前的初始化工作：初始化SimStruct结构，其中包含了关于S函数的相关信息；设置输入输出端口的数目和大小；设置模块的采样周期；分配内存并设置sizes数组。
- 下一个采样时刻的计算：如果设置的是可变步长的积分算法，则计算下一个计算时刻，如果是固定步长，则计算下一次仿真的步长。
- 在最大时间步长内计算模块的输出，这一步完成以后，模块所有的输出都变成有效值。
- 在最小时间步长内更新离散状态。
- 积分过程，这一步一般是针对连续系统而言。

下面我们介绍一下有关C MEX文件的情况。

在M文件的S函数中，S函数是通过M子函数完成的，而在C MEX文件中则是通过C子程序完成的，同时，SIMULINK还为C MEX提供了大量的M子函数。在M文件形式的S函数中，SIMULINK将一个flag的参数传递给S函数，该参数存储了目前仿真的状态，我们必须为每一个flag的值编写适当的函数。而对于C MEX文件，SIMULINK则直接调用相应的S子函数。下面列举了仿真的不同状态下flag的具体值以及调用的S函数。

仿真状态	S函数子程序	Flag值
初始化(Initialization)	mdlInitializeSizes	Flag = 0

计算下一个采样时刻(可选)	mdlGetTimeOfNextVarHit	Flag = 4
计算输出(Calculation of outputs)	mdlOutputs	Flag = 3
更新离散状态(Update discrete states)	mdlUpdate	Flag = 2
导数的计算(Calculation of derivatives)	mdlDerivatives	Flag = 1
仿真结束(End of simulation tasks)	mdlTerminate	Flag = 9

为了方便用户编写 S 函数，MATLAB 提供了一个 M 文件的模板 sfuntmpl.m，该文件放在 matlab/toolbox/SIMULINK/blocks 目录下，该模板采用 switch 处理不同的 flag 值，用户只要在相应的位置写入合适的代码。

对于 C MEX 文件形式的 S 函数，MATLAB 也提供了相应的模板 sfuntmpl.c，存放在 SIMULINK/src 目录下。其他信息可查看同一目录下的 sfuntmpl.doc 文档。

> 说明：由于 S 函数的编写比较复杂，而且不易调试，作者推荐使用相应的模板，这样可尽可能避免不必要的错误。

11.1.4　S 函数的基本概念

这一节我们主要介绍 S 函数中的一些基本概念，这些概念包括：直通（Direct feedthrough）、动态输入（Dynamically sized inputs）以及采样时间和间隔的设置（Setting sample times and offsets）。下面分别进行介绍：

1. 直通

直通意味着输出或可变采样时间直接受输入信号的控制。下列情况下将会发生直通：

- 输出函数中的参数包含输入 u。
- 下一采样时刻的计算需要输入 u。

例如：比较下面的系统：

$$y = k \times u \tag{11.2}$$

$$y = x, \dot{x} = u \tag{11.3}$$

其中，y 是输出，k 是增益，u 是输入信号。

从定义可以知道，系统（11.2）存在直通，而系统（11.3）不存在直通。

正确判断模块中有无直通现象相当重要，因为它影响到模块的执行顺序以及代数环的发生。

2. 动态输入

S 函数可以动态设置输入的向量宽度，在这种情况下，实际输入信号的宽度是由仿真开始时输入信号的宽度决定的，输入信号的宽度又可用来设置连续、离散状态和输出信号的数目。

在 M 文件中设置动态输入的方法是指定 mdlInitializeSizes 函数返回值为-1，可以采用 length(u)指令来得到实际的输入宽度。如果输入宽度为零，则模块中将不包含输入端口。在 C MEX 文件中的处理过程是相似的。

下面看一个例子（如图 11.1 所示）：

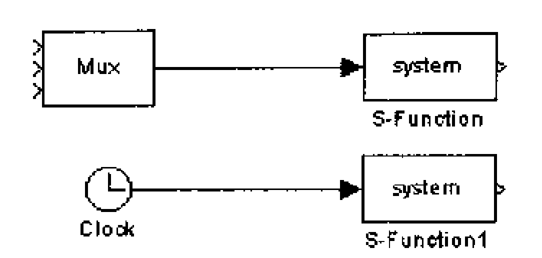


图 11.1　动态输入的 S 函数模块

图中上面的 S 函数模块是由三个信号共同驱动，而下面的 S 函数模块只由一个时钟信号驱动。由于在 S 函数中设置了动态输入，该模块会自动处理不同输入宽度的情况。同样，如果该模块的输出或状态变量也设置成动态的，则 S 函数将自动将它们设置成与实际输入相同的宽度。

3. 采样时间的设置

无论是 M 文件还是 C MEX 文件都为 S 函数中采样时间的设置提供了极大的灵活性。具体的采样时间的设置种类包括：

连续采样周期：一般针对连续系统。

连续但最小采样步长内不变的采样周期：在最大采样步长内计算正常进行，而在最小采样步长内所有的值保持不变。

离散采样周期：针对离散系统，同时可以设置采样间隔之间的延时量。

可变采样周期：S 函数中的采样周期是可变的。

继承性的采样周期：本身没有采样周期，具体时间是由其他模块继承而来的，这里其他模块可以包括前面的驱动模块，目标模块或模型中的最小采样周期。最典型的例子就是增益（Gain）模块，它的实际采样周期由它的前一个模块决定。

S 函数还支持多速率系统，也就是在同一个 S 函数中存在多个不同的采样周期。

为了了解以上概念的具体实现，读者可以查阅相应的模块源程序，SIMULINK 为此提供了大量的例子，它们都放置在指定目录下：

M-files: toolbox/SIMULINK/blocks

C MEX-files:SIMULINK/src

11.2　采用 M 文件编写 S 函数

11.2.1　S 函数模块的创建

M 文件形式的 S 函数是由一系列的 S 函数子程序构成。这些子程序在仿真过程中供 SIMULINK 本身调用，采用 M 文件编写 S 函数就是用自己的算法实现这些子程序，而 S 函数自身怎样同 SIMULINK 进行交互，用户可以不必过多深究，只需要按照 MAtLAB 提供的模板进行编写。

下面列举在编写 S 函数中所要遇到的子程序：

mdlInitializesizes	定义模块的基本属性，包括采样时间、连续或离散状态的初始值，定义 sizes 数组等
mdlDerivatives	计算连续状态变量的导数
mdlUpdate	根据需要更新离散状态变量、采样时间和最大步长
mdlOutputs	计算 S 函数的输出
mdlGetTimeOfNextVarHit	计算下一次计算的绝对时间，该函数只在可变离散采样时间（在 mdlInitializesizes 中进行设置）的条件下起作用
mdlTerminate	仿真结束时的工作

S 函数的创建可以通过两个步骤进行：

初始化模块的基本属性，例如输入输出信号的个数、采样时间的大小、各种状态变量的初始值等等。

在各种标准子程序中放置自己的算法。

1. 定义模块的基本属性

为了让 SIMULINK 找到该模块的基本属性，mdlInitializeSizes 函数首先调用 simsizes：

sizes = simsizes;

它返回一个空的 sizes 结构，该结构包含模块的基本属性，如下所示：

sizes.NumContStates	连续状态的数目
sizes.NumDiscStates	离散状态的数目
sizes.NumOutputs	输出端口的数目
sizes.NumInputs	输入端口的数目
sizes.DirFeedthrough	是否直通的标志
sizes.NumSampleTimes	采用周期的数目

用户首先填写该结构，然后再次调用simsizes：

sys = simsizes(sizes);

SIMULINK将根据sys向量得到该模块的基本属性。

2. 一个简单的例子

理解S函数机制的最好方法是多看各种模块的源程序。图 11.2 显示的是一个简单的例子，该系统仅由正弦函数模块、S 函数模块以及示波器模块组成。源程序记录在 timestwo.m 中。

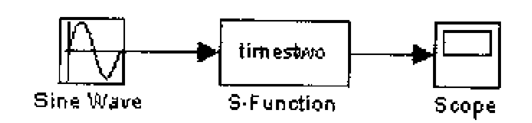


图 11.2　包含 S 函数的简单例子

```
function [sys, x0, str, ts] = timestwo(t, x, u, flag)
% 采用 switch 结构处理不同的 flag 值，对应于不同的仿真状态。
switch flag,
    case 0
```

```
        [sys,x0,str,ts] = mdlInitializeSizes; %初始化模块
    case 3
        sys = mdlOutputs(t,x,u); % 计算输出
    case { 1, 2, 4, 9 }
        sys = []; % 无用的标志
    otherwise
        error(['Unhandled flag = ',num2str(flag)]); % Error handling
    end;
     % 模块结束
```

从源程序角度来看，SIMULINK 将 t, x, u, flag 四个参数传递给 S 函数，同时需要 S 函数返回 sys, x0, str, ts 等四个向量，它们的具体含义是：

t	目前仿真中的实际时间
x	状态向量，可能为空
u	输入向量
flag	表明SIMULINK不同仿真状态的标志量
sys	依照flag 的不同返回不同的结果，如flag = 3，sys 返回S函数的输出
x0	初始状态值，如果没有状态变量，则为空向量，在flag = 0时忽略
str	保留为将来使用，必须设置为空矩阵
ts	两列矩阵，记录采样时间和模块的延时时间

以下是各个子程序的源代码。

```
%=============================================================
% mdlInitializeSizes 函数，初始化状态、采样周期、Size 结构
%=============================================================
function [sys,x0,str,ts] = mdlInitializeSizes
% 调用 simsizes 创建 sizes 结构
sizes = simsizes;
% 填写 sizes 结构
sizes.NumContStates= 0;
sizes.NumDiscStates= 0;
sizes.NumOutputs= 1;
sizes.NumInputs= 1;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
% Load the sys vector with the sizes information.
sys = simsizes(sizes);
%
x0 = []; % 没有连续状态
%
str = []; % 没有状态次序
%
ts = [-1 0]; % 继承性的采样周期
% mdlInitializeSizes 函数结束
%=============================================================
% mdlOutputs 计算模块输出
%=============================================================
```

```
function sys = mdlOutputs(t,x,u)
sys = 2*u;
% mdlOutputs 函数结束
```

将程序存盘，下面我们可以考察这个S函数的工作情况。双击S函数模块，在弹出的模块对话框中填写与这个S函数模块相关联的M文件名timestwo.m。运行仿真，看看仿真结果是否与预计的情况相同。

下面我们将针对不同类型的系统列举一个实例来说明这类系统的一般编写方法。

11.2.2 连续系统的S函数

下面为连续系统csfunc的源代码。

```
function [sys,x0,str,ts] = csfunc(t,x,u,flag)
% CSFUNC 连续系统 S 函数实例
% x' = Ax + Bu
% y = Cx + Du
%
%生成连续线性系统
A=[-0.09 -0.01
     1      0];
B=[ 1     -7
    0     -2];
C=[ 0      2
    1     -5];
D=[-3      0
    1      0];
switch flag,
    case 0
        [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D); %初始化模块
    case 1
        sys = mdlDerivatives(t,x,u,A,B,C,D); % 计算变量
    case 3
        sys = mdlOutputs(t,x,u,A,B,C,D); % 计算输出
    case { 2, 4, 9 } % 无用标志
        sys = [];
    otherwise
        error(['Unhandled flag = ',num2str(flag)]); % 错误处理
    end
     % csfunc 函数结束
%==============================================================
% mdlInitializeSizes 子函数
%==============================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)
%
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
```

```
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = zeros(2,1);    % 初始化状态变量
%
str = [];
%数组初始化
ts = [0 0];
% mdlInitializeSizes 子函数结束
%
%==============================================================
% mdlDerivatives
% 返回连续状态变量
%==============================================================
function sys = mdlDerivatives(t,x,u,A,B,C,D)
sys = A*x + B*u;
% mdlDerivatives 子函数结束
%==============================================================
% mdlOutputs 子函数
%==============================================================
%
function sys = mdlOutputs(t,x,u,A,B,C,D)
sys = C*x + D*u;
% mdlOutputs 子函数结束
```

这个系统与 timestwo 模块不同，它增加了 mdlDerivatives 子函数来计算 flag=1 时连续状态的导数。这个 S 函数描述的对象可以是任何可以用状态空间描述的连续系统。它完成的功能与 SIMULINK 当中标准的 State-Space 模块类似，从这个 S 函数出发，我们还可以进一步实现状态时变连续系统的 S 函数。

11.2.3　离散系统的 S 函数

```
function [sys,x0,str,ts] = dsfunc(t,x,u,flag)
% 定义离散系统的 S 函数实例
% 此 S 函数遵循下面的离散方程
% x(n+1) = Ax(n) + Bu(n)
% y(n) = Cx(n) + Du(n)
%
% 生成离散线性系统
A=[-1.3839    -0.5097
     1.0000      0];
B=[-2.5559      0
       0      4.2382];
C=[ 0         2.0761
    0         7.7891];
D=[ -0.8141   -2.9334
```

```
      1.2426          0];
  switch flag,
    case 0
        sys = mdlInitializeSizes(A,B,C,D); % 初始化
    case 2
        sys = mdlUpdate(t,x,u,A,B,C,D); % 提升离散状态
    case 3
        sys = mdlOutputs(t,x,u,A,B,C,D); % 计算输出
    case {1, 4, 9}     % 无用标志
        sys = [];
    otherwise
        error(['unhandled flag = ', num2str(flag)]);    % 错误处理
end
%  dsfunc.子函数结束
%==============================================================
%初始化
%==============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)
% Call simsizes for a sizes structure, fill it in, and convert it
% to a sizes array.
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 2;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1; % 矩阵非空
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = ones(2,1);      % 初始化离散状态
str = []; % 设 str 为零矩阵
ts = [1 0]; %
% mdlInitializeSizes.子函数结束
%==============================================================
%提升离散状态
%==============================================================
function sys = mdlUpdates(t,x,u,A,B,C,D)
sys = A*x + B*u;
% mdlUpdate.子函数结束
%==============================================================
% 计算输出
%==============================================================
function sys = mdlOutputs(t,x,u,A,B,C,D)
sys = C*x + D*u;
% mdlOutputs.子函数结束
```

上面的例子将完成状态空间形式的离散系统，系统的离散状态方程表示为

$$x(n+1) = Ax(n) + Bu(n)$$

$$y(n) = Cx(n) + Du(n)$$

因此，只要离散系统可以用上面的状态方程来表示，就可以通过该S函数来实现。其功能与SIMULINK内建的Discrete State-Space模块相似。可以将dsfunc.m文件作为建模时变离散状态系统的模板。

11.2.4　混合系统的S函数

混合系统是指系统中既包含连续状态又包含离散状态。SIMULINK根据flag的具体值分别为系统的连续或离散部分调用相应的子程序。SIMULINK在处理混合系统时将同时调用S函数中的mdlUpdate、mdlOutput和mdlGetTimeOfNextVarHit子程序。因此用户在写相应的代码时需要注意只处理当前仿真时刻的数据。

SIMULINK本身包含了一个混合系统的例子，文件名为mixedm.m，相应的系统方框图可用图11.3表示。

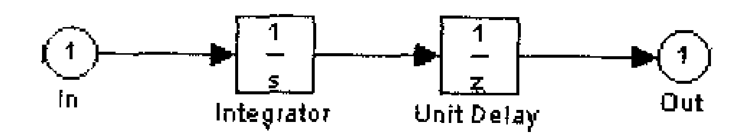


图11.3　混合系统的方框图

以下是 mixedm.m 中实现的代码：

```
function [sys,x0,str,ts] = mixedm(t,x,u,flag)
% 混合系统的例子，由一个积分器和一个单位时延模块串联而成。
% 设置积分器的采样周期和时延模块的延迟时间
dperiod = 1;
doffset = 0;
switch flag,
        case 0                                          %初始化
              [sys,x0,str,ts] = mdlInitializeSizes(dperiod,doffset);
        case 1
              sys = mdlDerivatives(t,x,u);              % 计算导数
        case 2
              sys = mdlUpdate(t,x,u,dperiod,doffset); %更新状态
        case 3
              sys = mdlOutputs(t,x,u,doffset,dperiod); % 计算输出
        case {4, 9}
              sys = [];                                 % 无用的标志
        otherwise
            error(['unhandled flag = ',num2str(flag)]); %错误处理
end
% mixedm 函数结束
%
%=============================================================
% mdlInitializeSizes
%=============================================================
```

```
function [sys,x0,str,ts] = mdlInitializeSizes(dperiod,doffset)
sizes = simsizes;
sizes.NumContStates = 1;
sizes.NumDiscStates = 1;
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 2;
sys = simsizes(sizes);
x0 = ones(2,1);
str = [];
ts = [0, 0 % 采样时间
dperiod, doffset];
% mdlInitializeSizes 结束
%
%==============================================================
% mdlDerivatives，计算连续变量的导数
%==============================================================
%
function sys = mdlDerivatives(t,x,u)
sys = u;
% mdlDerivatives 结束
%==============================================================
% mdlUpdate
%==============================================================
%
function sys = mdlUpdate(t,x,u,dperiod,doffset)
% 下一个离散状态是积分器的输出，如果误差在容许误差 1e-8 内，就返回下一个离散状态，%如果不在采样时刻则返回空表示离散状态不发生变化
if abs(round((t-doffset)/dperiod)-(t-doffset)/dperiod) < 1e-8
        sys = x(1);
else
        sys = [];
end
% mdlUpdate 子函数结束
%
%==============================================================
% mdlOutputs 子函数
%==============================================================
%如果误差在容许误差 1e-8 内，就返回下延时模块的输出，
%如果不在采样时刻则返回空表示输出状态不发生变化
if abs(round((t-doffset)/dperiod)-(t-doffset)/dperiod) < 1e-8
      sys = x(2);
else
      sys = [];
end
% mdlOutputs 子函数结束
```

11.3　采用 C MEX 文件编写 S 函数

11.3.1　概述

C MEX文件形式的S函数与M形式的S函数在函数结构和子函数功能上基本相同，但C语言本身的灵活性使得C MEX相比M文件在算法实现上具有更强的功能。Simulink同样为C MEX文件的编写提供了模板sfuntmpl.c，更详细的内容可参考文档sfuntmpl.doc。C MEX 文件形式的S函数具有下面的一般结构：

```
#define S_FUNCTION_NAME    your_sfunction_name_here
 #define S_FUNCTION_LEVEL 2
 #include "simstruc.h"
static void mdlInitializeSizes(SimStruct *S)
{
}
<加入S函数的实现子程序和相关代码>
static void mdlTerminate(SimStruct *S)
 {
}
#ifdef MATLAB_MEX_FILE   /*该文件是否作为一个 MEX文件被编译？  */
#include "simulink.c"   /* MEX文件的接口文件*/
#else
#include "cg_sfun.h" /* 代码生成的注册函数，用于对Real-Time Workshop的连接*/
#endif
```

mdlInitializeSizes 是 Simulink 与 S 函数交互时调用的第一个子程序，S 函数的其他子程序都是以 mdl 开头。mdlTerminate 是 Simulink 调用的最后一个子程序。与 M 文件形式的 S 函数不同，这里没有 flag 标识参数来表明 Simulink 的仿真状态，Simulink 对各个子程序的调用是自动进行的。Simulink 在一个称为 SimStruct 的结构中维护模块的基本属性，该结构在头文件 simstruc.h 中定义，因此，在每个 C MEX 的开始，都必须将该头文件包含进来。该头文件除了定义结构 SimStruct 外，还定义了一系列对 SimStruct 结构进行操作的宏。

11.3.2　简单 C MEX 文件的创建

再来看看 11.2.1 节讨论的例子。该系统的 C MEX 文件实现的源程序位于 matlab/simulink/src/timestwo.c 中。

为了将该函数包含进 Simulink 中，在命令窗口中输入指令：

```
>> mex timestwo.c
```

MATLAB 将该系统编译成 MEX 文件，MEX 是“MATLAB Executable”的缩写形式，其文件扩展名随操作系统不同而有所变化，在 Microsoft Windows 中扩展名为.dll。

以下是源程序代码：

```
#define S_FUNCTION_NAME    timestwo
#define S_FUNCTION_LEVEL   2
```

```
#include "simstruc.h"
static void mdlInitializeSizes(SimStruct *S)
{
ssSetNumSFcnParams(S, 0);
if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
{
    return;    /*将由SIMULINK报告该函数的输入参数不匹配*/
}
if (!ssSetNumInputPorts(S, 1))
    return;
ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
ssSetInputPortDirectFeedThrough(S, 0, 1);
if (!ssSetNumOutputPorts(S,1))
    return;
ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);
ssSetNumSampleTimes(S, 1);
/* 设置以下内容时要小心，参考sfuntmpl.doc文件*/
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
static void mdlInitializeSampleTimes(SimStruct *S)
{
ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
ssSetOffsetTime(S, 0, 0.0);
}
static void mdlOutputs(SimStruct *S, int_T tid)
{
int_T i;
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
real_T *y = ssGetOutputPortRealSignal(S,0);
int_T width = ssGetOutputPortWidth(S,0);
for (i=0; i<width; i++)
{
    *y++ = 2.0 *(*uPtrs[i]);
}
}
/* 下面的函数尽管为空，但必不可少 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* 该文件是否作为一个 MEX文件被编译？ */
#include "simulink.c"  /* MEX文件的接口文件 */
#else
#include "cg_sfun.h" /* 代码生成的注册函数，用于对Real-Time Workshop的连接*/
#endif
```

MEX 文件形式的 S 函数对离散和连续状态的处理是分别在 mdlUpdate 和 mdlDerivative 子函数中进行的。通常，大多数 S 函数需要对状态（连续或离散）进行处理。在 SIMULINK 中用 S 函数可以建模的系统有：连续、离散、混合、变步长采样时间、过零区间、时间变化

连续传递函数。

下面的所有例子都是基于 C MEX 文件形式的 S 函数的模板文件 sfuntmpl.c 和 sfuntmpl.doc 生成的。

11.3.3 连续系统的 C MEX 实现

Simulink 自带的 csfunc.c 源程序演示了如何在 C MEX 的 S 函数中定义连续状态方程描述的连续系统，以下是具体代码：

```
matlab/simulink/src/csfunc.c
/* 文件: csfunc.c
*连续系统的例子
* x' = Ax + Bu
* y = Cx + Du
*/
#define S_FUNCTION_NAME csfunc
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
     #define U(element) (*uPtrs[element]) /* 指向输入端口Port0的指针*/
static real_T A[2][2]={
                          { -0.09, -0.01 } ,
                          { 1 , 0        }
                        };
static real_T B[2][2]={
                          { 1 , -7 } ,
                          { 0 , -2 }
                        };
static real_T C[2][2]={
                          { 0 , 2 } ,
                          { 1 , -5 }
                        };
     static real_T D[2][2]={
                          { -3 , 0 } ,
                          { 1 , 0 }
                        };
/*=====================*
* S函数的子函数 *
*=====================*/
/* 子函数: mdlInitializeSizes
*  提示:
*  Sizes信息被SIMULINK用于确定S函数模块的特征（输入、输出以及状态的个数）
*/
static void mdlInitializeSizes(SimStruct *S)
{
          ssSetNumSFcnParams(S, 0);    /* 输入的参数个数 */
         if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
```

```
            {
               return;      /* 将由SIMULINK报告该函数的输入参数不匹配*/
            }
 ssSetNumContStates(S, 2);
 ssSetNumDiscStates(S, 0);
    if (!ssSetNumInputPorts(S, 1))
         return;
 ssSetInputPortWidth(S, 0, 2);
 ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1))
        return;
 ssSetOutputPortWidth(S, 0, 2);
 ssSetNumSampleTimes(S, 1);
 ssSetNumRWork(S, 0);
 ssSetNumIWork(S, 0);
 ssSetNumPWork(S, 0);
 ssSetNumModes(S, 0);
 ssSetNumNonsampledZCs(S, 0);
 /*  设置以下内容时要小心，参考sfuntmpl.doc文件 */
 ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
 }
 /* 子函数: mdlInitializeSampleTimes

 * 提示:
 * 指定我们将需要一个连续的采样时间
 */
 static void mdlInitializeSampleTimes(SimStruct *S)
 {
 ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
 ssSetOffsetTime(S, 0, 0.0);
 }
 #define MDL_INITIALIZE_CONDITIONS
 /* 子函数: mdlInitializeConditions
 * 提示:
 * 将两个连续状态初始化为0
 */
 static void mdlInitializeConditions(SimStruct *S)
 {
 real_T *x0 = ssGetContStates(S);
 int_T lp;
 for (lp=0;lp<2;lp++)
 {
        *x0++=0.0;
          }
 }
/*子函数: mdlOutputs
```

```
* 提示:
* y = Cx + Du
*/
static void mdlOutputs(SimStruct *S, int_T tid)
{
real_T *y = ssGetOutputPortRealSignal(S,0);
real_T *x = ssGetContStates(S);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
/* y=Cx+Du */
y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
}
#define MDL_DERIVATIVES
/*子函数: mdlDerivatives
* 提示:
* xdot = Ax + Bu
*/
static void mdlDerivatives(SimStruct *S)
{
real_T *dx = ssGetdX(S);
real_T *x = ssGetContStates(S);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
/* xdot=Ax+Bu */
dx[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*U(1);
dx[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*U(1);
}
/*子函数:  mdlTerminate
* 提示:
* 尽管该函数这里不作任何事, 但必不可少
*/
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* 该文件是否作为一个 MEX文件被编译? */
#include "simulink.c"   /* MEX文件的接口文件 */
#else
#include "cg_sfun.h" /* 代码生成的注册函数, 用于对Real-Time Workshop的连接*/
#endif
```

该文件是 csfunc.m 文件的 C 语言版本。状态向量 x 的导数的计算是在 mdlDerivative 函数中，输出 y 的计算是在 mdlOutputs 函数中。因为没有离散状态并且在结束时没有任何任务需要完成，所有没有 mdlUpdate 子函数，并且 mdlTerminate 子函数是空的。

11.3.4　离散系统的 C MEX 实现

SIMULINK 包含一个名为 dsfunc.c 的函数，它是一个用 S 函数建模的离散状态系统。下面是该 S 函数的 C 语言代码：

```
matlab/simulink/src/dsfunc.c
/* 文件: dsfunc.c
 * 提示:
 *
 * 定义离散系统的 C MEX 文件形式的S函数
 *
 * x(n+1) = Ax(n) + Bu(n)
 * y(n) = Cx(n) + Du(n)
    *
 * 如果要了解更为详细的内容，可以参考simulink/src/sfuntmpl.doc文档
 *
 */
#define S_FUNCTION_NAME dsfunc
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element]) /* 指向输入端口Port0 */
static real_T A[2][2]={
                        { -1.3839, -0.5097 } ,
                        { 1 , 0 }
                      };
static real_T B[2][2]={
                        { -2.5559, 0 } ,
                        { 0 , 4.2382 }
                      };
static real_T C[2][2]={
                        { 0 , 2.0761 } ,
                        { 0 , 7.7891 }
                      };
static real_T D[2][2]={
                        { -0.8141, -2.9334 } ,
                            { 1.2426, 0 }
                      };
/*=====================*
 * S函数的子函数*
 *=====================*/
/* 子函数: mdlInitializeSizes
 * 提示:
 *  Sizes信息被SIMULINK用于确定S函数模块的特征（输入、输出以及状态的个数）
 */
static void mdlInitializeSizes(SimStruct *S)
{
```

```
ssSetNumSFcnParams(S, 0); /* 输入的参数个数 */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
     {
        return; /* 将由SIMULINK报告该函数的输入参数不匹配*/
     }
ssSetNumContStates(S, 0);
ssSetNumDiscStates(S, 2);
  if (!ssSetNumInputPorts(S, 1))
      return;
ssSetInputPortWidth(S, 0, 2);
ssSetInputPortDirectFeedThrough(S, 0, 1);
  if (!ssSetNumOutputPorts(S, 1))
      return;
ssSetOutputPortWidth(S, 0, 2);
ssSetNumSampleTimes(S, 1);
ssSetNumRWork(S, 0);
ssSetNumIWork(S, 0);
ssSetNumPWork(S, 0);
ssSetNumModes(S, 0);
ssSetNumNonsampledZCs(S, 0);
/*  设置以下内容时要小心，参考sfuntmpl.doc文件 */
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
/* 子函数: mdlInitializeSampleTimes
* 提示:
* 设置采样时间的属性，这里设置成由驱动模块继承
*/
static void mdlInitializeSampleTimes(SimStruct *S)
{
ssSetSampleTime(S, 0, 1.0);
ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
    /* 子函数: mdlInitializeConditions
* 提示:
* 将两个连续状态初始化为0
*/
static void mdlInitializeConditions(SimStruct *S)
{
real_T *x0 = ssGetRealDiscStates(S);
int_T lp;
for (lp=0;lp<2;lp++) {
*x0++=1.0;
}
    }
/*子函数:  mdlOutputs
```

```
* Abstract:
* y = Cx + Du
*/
static void mdlOutputs(SimStruct *S, int_T tid)
{
real_T *y = ssGetOutputPortRealSignal(S,0);
real_T *x = ssGetRealDiscStates(S);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* y=Cx+Du */
y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
}
#define MDL_UPDATE
/*子函数: mdlUpdate
* 提示:
* xdot = Ax + Bu
*/
static void mdlUpdate(SimStruct *S, int_T tid)
{
real_T tempX[2] = {0.0, 0.0};
real_T *x = ssGetRealDiscStates(S);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
/* xdot=Ax+Bu */
tempX[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*U(1);
tempX[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*U(1);
x[0]=tempX[0];
x[1]=tempX[1];
}
/*子函数: mdlTerminate
* 提示:
* 尽管该函数这里不作任何事，但必不可少
*/
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* 该文件是否作为一个 MEX文件被编译? */
#include "simulink.c"   /* MEX文件的接口文件 */
#else
#include "cg_sfun.h" /* 代码生成的注册函数，用于对Real-Time Workshop的连接*/
#endif
```

上面是 dsfunc.m 的 C 语言的等效形式。对离散状态向量 x 的更新是在 mdlUpdate 函数中，对输出的计算是在 mdlOutputs 函数中。因为没有连续状态并且在结束时没有任务需要完成，所有没有 mdlDerivatives 子函数，并且 mdlTerminate 是空函数。

11.3.5　混合系统的 C MEX 实现

SIMULINK 包含一个名为 mixdem.c 的函数，它是一个用 S 函数 mixedm.c 建模的混合系统（同时包含连续和离散状态）的例子。mixedm.c 组合了 csfunc.c 和 defunc.c 中元素。如果需要对一个混合模型建模，所要做的就是在 mdlDerivative 中放入该模型的连续方程，在 mdlUpdate 中放进模型的离散方程。下面是用 C 语言编写的混合系统的 S 函数，保存于 mixedm.c 中，它实现了一个连续的积分器后紧跟一个离散的单位延迟，具体代码如下：

```
matlab/simulink/src/mixedm.c
/* 文件: mixedm.c
 * 提示:
 * 该S函数实现了一个连续的积分器（1/s）后紧跟一个离散的单位延迟（1/z）
 *
 *更为详细的说明可以参考 simulink/src/sfuntmpl.doc文档
 */
#define S_FUNCTION_NAME mixedm
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element]) /* 指向输入端口Port0 */
/*======================*
 * S函数子函数 *
 *======================*/
/* 子函数: mdlInitializeSizes
 *  提示:
 *  Sizes信息被SIMULINK用于确定S函数模块的特征（输入、输出以及状态的个数）
 */
static void mdlInitializeSizes(SimStruct *S)
{
ssSetNumSFcnParams(S, 0); /* 输入的参数个数 */
if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
 {
        return;  /* 将由SIMULINK报告该函数的输入参数不匹配*/
 }
ssSetNumContStates(S, 1);
ssSetNumDiscStates(S, 1);
if (!ssSetNumInputPorts(S, 1)) return;
ssSetInputPortWidth(S, 0, 1);
ssSetInputPortDirectFeedThrough(S, 0, 1);
if (!ssSetNumOutputPorts(S, 1)) return;
ssSetOutputPortWidth(S, 0, 1);
ssSetNumSampleTimes(S, 2);
ssSetNumRWork(S, 0);
ssSetNumIWork(S, 0);
ssSetNumPWork(S, 0);
ssSetNumModes(S, 0);
```

```
ssSetNumNonsampledZCs(S, 0);
/* 设置以下内容时要小心，参考sfuntmpl.doc文件 */
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
/* 子函数: mdlInitializeSampleTimes
*  提示:
*  设置两个采样时间，一个为连续的，一个周期为1.0秒的离散采样时间
*/
static void mdlInitializeSampleTimes(SimStruct *S)
{
ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
ssSetSampleTime(S, 1, 1.0);
ssSetOffsetTime(S, 0, 0.0);
ssSetOffsetTime(S, 1, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
/*子函数: mdlInitializeConditions
* 提示:
* 将两个连续状态初始化为0
*/
static void mdlInitializeConditions(SimStruct *S)
{
real_T *xC0 = ssGetContStates(S);
real_T *xD0 = ssGetRealDiscStates(S);
    xC0[0] = 1.0;
xD0[0] = 1.0;
}
/*子函数:  mdlOutputs
* Abstract:
* y = xD
*/
static void mdlOutputs(SimStruct *S, int_T tid)
{
real_T *y = ssGetOutputPortRealSignal(S,0);
real_T *xD = ssGetRealDiscStates(S);
/* y=xD */
if (ssIsSampleHit(S, 1, tid))
{
  y[0]=xD[0];
}
}
#define MDL_UPDATE
/*子函数:  mdlUpdate
* Abstract:
* xD = xC
```

```
*/
static void mdlUpdate(SimStruct *S, int_T tid)
{
real_T *xD = ssGetRealDiscStates(S);
real_T *xC = ssGetContStates(S);
/* xD=xC */
if (ssIsSampleHit(S, 1, tid))
{
   xD[0]=xC[0];
}
}
#define MDL_DERIVATIVES
/* 子函数: mdlDerivatives
* Abstract:
* xdot = U
*/
static void mdlDerivatives(SimStruct *S)
{
real_T *dx = ssGetdX(S);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
/* xdot=U */
dx[0]=U(0);
}
/*子函数: mdlTerminate
* 提示:
* 尽管该函数这里不作任何事，但必不可少
*/
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* 该文件是否作为一个 MEX文件被编译? */
#include "simulink.c"  /* MEX文件的接口文件 */
#else
#include "cg_sfun.h" /* 代码生成的注册函数，用于对Real-Time Workshop的连接*/
#endif
```

因为在结束时没有任务需要完成，所以 mdlTerminate 是一个空函数。MdlDerivative 计算连续状态向量 x 的导数，而 mdlUpdate 包含有用来更新离散状态 x 的方程。

由于篇幅所限，这里只列举了连续状态、离散状态和混合状态系统的 S 函数实现，一些比较特殊的系统实现，如变步长的 S 函数、过零函数以及条件子系统的实现方法，读者可以参考相关的帮助文档和模板文件。

总之，S 函数是 SIMULINK 当中惟一需要编程的地方，但它为用户扩展 SIMULINK 的基本模块提供了灵活的手段。尽管 S 函数的编写相对复杂，但 SIMULINK 本身提供的丰富的模块库一般就能满足用户绝大多数的需求。用户在建模过程中，能够用标准模块实现的就尽量采用标准模块实现。而对于 S 函数，只要用户平时加以练习，也能很快掌握其编写方法。

第 12 章　控制系统的建模与仿真

控制系统计算机辅助设计是一门以计算机为工具进行的控制系统设计与分析的技术。

控制系统计算机辅助设计的软件包是从 20 世纪 70 年代发展起来的，最初是由英国的 H. H. Rosenbrock 学派将线性单变量控制系统的频域理论推广到多变量系统，随后 Manchester 大学的控制系统中心完成了该系统的计算机辅助设计软件包；日本的古田胜久主持开发的 DPACS-F 软件，在处理多变量系统的分析和设计上也很有特色。同时，国际自动控制领域的一些学者用状态空间法发展了多变量系统控制理论，在控制系统设计软件上提供了命令式的人机交互界面，在控制系统设计与仿真中给设计者以充分的主动权。

20 世纪 80 年代以来，在从多仿真语言和仿真软件包的中，MATLAB 以其模块化的计算方法，可视化与智能化的人机交互功能，丰富的矩阵运算、图形绘制、数据处理函数以及模块化图形组态的动态系统仿真工具 SIMULINK，成为控制系统设计仿真领域最受欢迎的软件系统。

本章就经典控制理论和现代控制理论中的一些具体问题为研究对象，介绍了 SIMULINK 在控制系统设计和仿真中的具体使用方法，从而使读者进一步体会 SIMULINK 软件在动态系统仿真中的独特魅力。

12.1　离散系统建模

12.1.1　离散系统建模的基本概念

（1）离散模块

每一个离散模块在其输入都有一个采样器，并且在其输出端都有零阶保持器。如果模型当中既有离散模块，又有连续模块，则离散模块的输出在两个采样时刻之间取常值。离散模块的输出只在自身的采样点处进行更新计算。

（2）采样时间

Sample time 参数用于设置离散模块的采样周期，这也是离散模块中状态变量的更新周期。一般情况下，该参数设置为标量，有的时候也设置成带两个参数的向量形式，增加的参数表示采样时间的偏移量。

例如，采样周期常常设置成[Ts，offset]的形式。Ts 表示采样时间（周期），offset 表示相应的时间偏移量。离散模块实际的更新时间是：

$$t = n * Ts + offset$$

偏移量在某些离散模块的采样时间需要设置成比其他离散模块要快或慢一些时特别有用。

用户不可以在仿真进行当中改变离散模块的采用时间，如果需要改变，就必须停止仿真的执行，待修改完成后，重新开始仿真。

（3）纯离散系统

全部由离散模块组成的系统模型称为纯离散系统（Purely Discrete Systems）。该离散系统的求解器可以设置成任何一种算法，而它们的仿真结果都相同。如果只在采样点处产生系统输出，则可以选择离散求解算法。

（4）多采样速率系统

多速率系统由具有不同的采样时间的离散模块和连续模块组成。例如考虑如图12.1显示的简单的多速率离散系统模型，DTF1离散传递函数模块的采样时间设置为[1 0.1]，偏移量为0.1。而DTF2离散传递函数模块的采样时间设置为0.7，没有偏移量。开始仿真，并使用stairs指令绘制输出。

```
>>[t,x,y] = sim('multirate', 3);
>>stairs(t,y)
```

输出结果如图12.2所示。

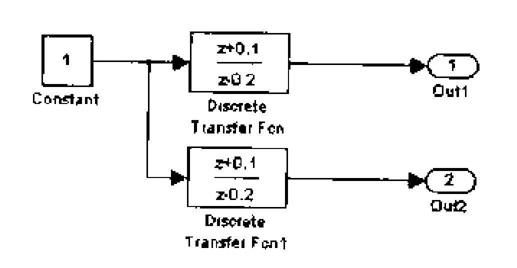


图 12.1　多采样速率离散系统模型

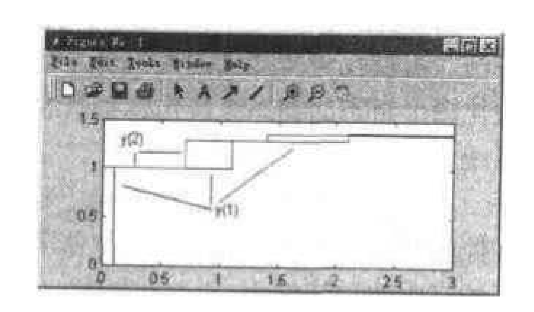


图 12.2　多采样速率离散系统的运行结果

从结果中可以看出，对于 DTF1 模块，直到 t = 0.1 时刻才产生输出，这是因为 DTF1 模块的初始条件是 0。

12.1.2　不同采样速率的彩色显示

SIMULINK可以使用不同的颜色区分模型当中的不同采样周期。

- 黑色：连续系统模块
- 品红色：常值模块
- 黄色：混合系统（包含不同采样速率的子系统或处理具有不同采样速率信号的Mux或Demux 模块
- 红色：表示最快的离散采样频率
- 绿色：表示次快的离散采样频率，其他从高到低依次为蓝色、淡蓝色和暗绿色
- 灰色：采用固定最小步长的采样频率

要使SIMULINK用彩色显示不同的采样频率，可以选择“Format：Sample Time Colors”菜单。

SIMULINK并不会自动更新时间颜色特性，因此必须选择“Edit：Update Diagram”菜单项以更新模型的颜色。要返回到原来的颜色，可以再选择“Format：Sample Time Colors”菜单。

如果使用采样时间颜色，分配给各个模块的颜色，取决于模型中其采样时间与其他模块的采样时间的比较。

SIMULINK根据下面的规则为单个模块设置采样时间。

连续模块（如Integrator、Derivative、Transfer Fcn等）是连续的；

常数模块（如Constant）是常数的；

离散模块（如Zero-Order Hold、Unit Delay、Discrete Transfer Fcn等等）可以让用户在其属性对话框中设置具体的采样时间；

所有其他的模块，具有基于其输入采样时间的隐含指定的采样时间。例如，跟在Integrator模块后的Gain模块被看作是连续的；而跟在Zero-Order Hold模块后的Gain模块被看作是离散的，且它的采样时间与Zero-Order Hold模块相同。这种现象又称为由驱动模块继承。

对于那些各个输入具有不同采样时间的模块，如果所有采样时间是最快采样时间的整数倍，模块将被赋予最快输入的采样时间。如果使用了变步长的求解器，模块将被赋予连续采样时间，如果使用的是定步长求解器，并且采样时间的最大公约数（基本采样时间）能够计算出来，就将它作为模块的采样时间，否则使用连续采样时间。

Mux和Demux模块只是简单的信号组合与分解操作，传过它们的信号保持其时间信息。因此，从Demux模块发出的信号线（如果它们不是用不同采样时间的信号源驱动），可能具有不同的颜色。在这种情况下，Mux与Demux模块用（黄色）颜色标注为混合颜色，以表明它们处理多采样速率的信号。

同样，包含有不同采样速率模块的子系统模块，也用（黄色）颜色标注为混合模块。如果子系统中的所有模块运行于单一的采样速率，则Subsystem模块根据该采样速率着色。

12.1.3　混合系统建模

混合的连续和离散系统由离散的模块和连续的模块混合组成，这样的系统能够用各种综合方法进行仿真，其中某些方法会比另外的方法更有效、更精确。对于大多数混合的连续和离散系统，Runge-Kutta 变步长方法，ode23 和 ode45 方法在效率和精度方面优于其他的方法。由于不连续性与离散模块的采样和保持有关，因此在混合系统中，一般不要使用 ode15s 和 ode113 求解方法。

12.2　经典控制系统的设计与仿真

12.2.1　控制系统的时域分析方法

经典控制理论研究的对象是单输入单输出定常系统，一般用传递函数来描述控制系统，采用的分析和设计方法有根轨迹方法、时域和频域方法等。MATLAB 作为控制领域的著名设计和仿真软件，向用户提供了强大的控制系统设计和仿真手段。本节我们就以简单的二阶系统为例，来讨论怎样利用 MATLAB 及其 SIMULINK 软件包进行控制系统的设计与仿真。

时域方法是经典控制理论中的常用方法，也是一种十分古老的方法。这种方法主要利用

控制系统对阶跃信号的相应曲线来了解系统的动态特性。借助 SIMULINK，我们可以实时的分析并观察系统的阶跃响应曲线。

二阶系统的闭环传递函数可写成如下的标准形式

$$G(s)=\frac{\omega_n^2}{S^2+2\xi_n\omega_n S+\omega_n^2} \tag{12.1}$$

其中，ξ 为阻力尼比，ω_n 为无阻力情况下的振荡频率。随着阻尼比的不同，系统闭环极点的位置也不同，从而导致不同的系统特性：

- $0<\xi<1$，欠阻尼情况
- $\xi=1$，临界阻尼情况
- $\xi>1$，过阻尼情况
- $\xi=0$，无阻尼情况

分别取 $\xi=0,0.4,0.8,1.0,1.4,\omega_n=1$，二阶系统阶跃响应的仿真框图及仿真结果如图 12.3 和图 12.4 所示。

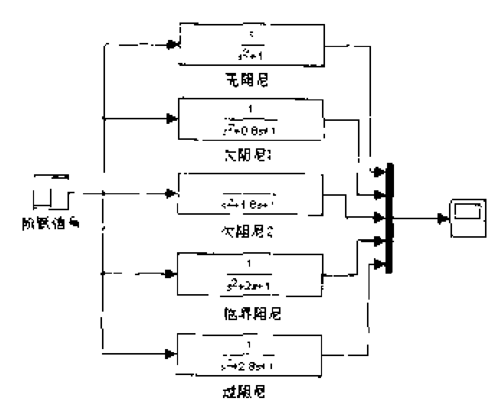


图 12.3　阶跃响应的仿真框图

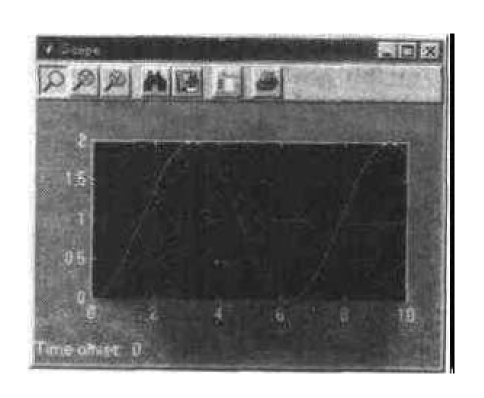


图 12.4　不同阻尼比下的仿真结果

从图中的仿真结果来看，二阶系统随着阻尼比的减小，振荡的幅度越来越大，在无阻尼情况下出现了等幅振荡，而在过阻尼情况，二阶系统的过渡曲线单调上升。

在 $\xi=0.4\sim0.8$ 时系统的过渡过程不仅具有较短的响应时间，而且振荡幅值也不大，这是二阶系统一般的理想工作状态。

12.2.2　控制系统的频域分析方法

频率分析方法是控制系统设计的重要方法，尤其在工程中得到广泛的应用。频率分析方法具有明确的物理意义，计算量小，一般可以采用作图的方法或实验的方法求出系统的频率特性，这对于实际工程中很难有明确模型表示的系统和元件，具有重要的实用价值。

由于系统的频率特性 $G(j\omega)$ 的代数表达式比较复杂，一般是采用图形表达方法。最常用的频率特性是伯特图。借助 MATLAB 中的 bode 指令，我们很容易绘制系统的伯特图，其使用方法是：

```
bode(num,den)
[mag, phase, w] = bode(num,den)
[mag, phase] = bode(num,den,w)
```

这里的 w 表示频率。第 1 种格式分别生成系统的幅频特性和相频特性，第 2 种格式可以

生成 1 行向量表示的频率点，第 3 个命令可以定义所显示的频率范围。借助 subplot 指令还可以调整伯特图的显示。

```
>>subplot(2,1,1)，semilogx(w,20*log10(mag))
>>subplot(2,1,1)，semilogx(w,phase)
```

若要指定系统的频率范围，可以使用 logspace 指令

```
>>w = logspace(m,n,npts)
```

上面的指令生成一个以 10 为底的对数向量，点数可任意选定。

另外，nyquist 和 nichols 指令还可以得到系统的 Nyquist 曲线和 Nichols 图。

例 12.1：**开环传递函数为 $G(s)=\dfrac{1.5}{s(s+1)(s+2)}$ 的控制系统，画出它的伯特图。**

```
>>k=1.5;ng=1.0;dg=poly([0-1-2]);
>>w=logspace(-1,1,-100);
>>[m,p] = bode(k*ng,dg,w);
>>subplot(2,1,1);semilogx(w,20*log10(m));
>>grid;ylabel('增益 dB');
>>subplot(2,1,2);semilogx(w,p);grid;
>>xlabel('频率 (rad/s)'); ylabel('相角 (deg)');
```

图 12.5 显示的是最终的执行结果。

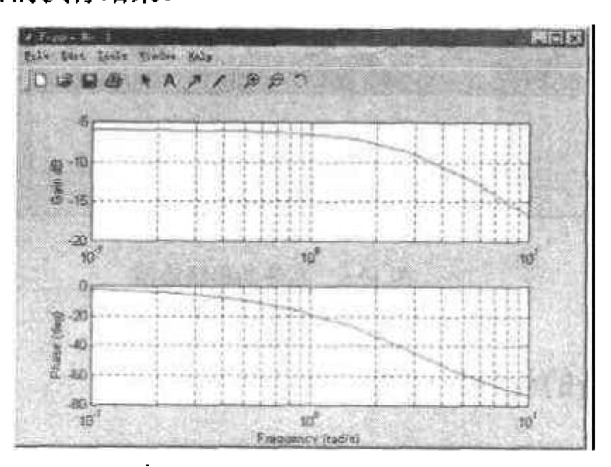


图 12.5　开环系统伯特图

12.2.3　控制系统的根轨迹分析方法

根轨迹方法是 W. R. Evans 提出的一种求解闭环系统特征值的简便方法。由于控制系统的动态特性是由系统闭环零极点共同决定的，而控制系统的稳定性又是由闭环系统极点的位置惟一决定的，因此，在分析控制系统动态性能时，确定闭环系统的零极点在 S 平面上的位置就可以了解系统的基本特性。

MATLAB 为绘制单输入单输出系统绘制根轨迹提供了图形界面的设计工具 rlocus，借助

该工具可以迅速完成根轨迹的绘制和系统零极点的设计。在命令窗口中输入：

```
>>rlocus(num,den)
>>rlocus(num,den,k)
```

将出现根轨迹的设计环境。

在系统的分析过程中，常常希望确定根轨迹上某一点处的增益，rlocfind 指令可以完成这一功能。

```
>>[k，poles] = rlocfind(num,den)
```

执行该指令后，图形中将生成一个十字光标，使用鼠标移动它到所希望的位置，然后单击左键即可得到该极点位置的坐标值和对应的增益值。

例 12.2：绘制例 12.1 系统的根轨迹图。

```
>>ng=1.0;
>>dg=plot([0,-1,-2]);
>>rlocus(ng,dg);
```

根轨迹图如图 12.6 所示。

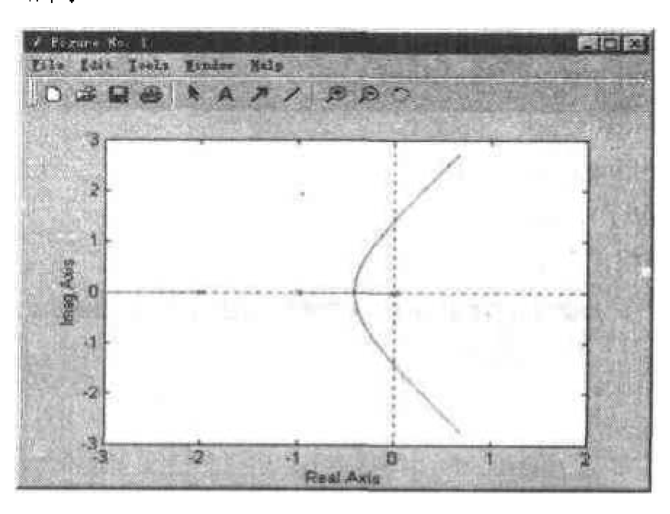


图 12.6　系统的根轨迹图

12.2.4　常用控制器的设计与仿真

用经典控制理论进行控制系统设计就是在系统中引入适当的控制环节，使系统的动态特性满足规定的性能指标要求，也被称为系统的校正与综合。按照控制环节频域特性的不同，可分为超前校正环节、滞后校正环节以及滞后-超前校正环节。

1. 超前校正

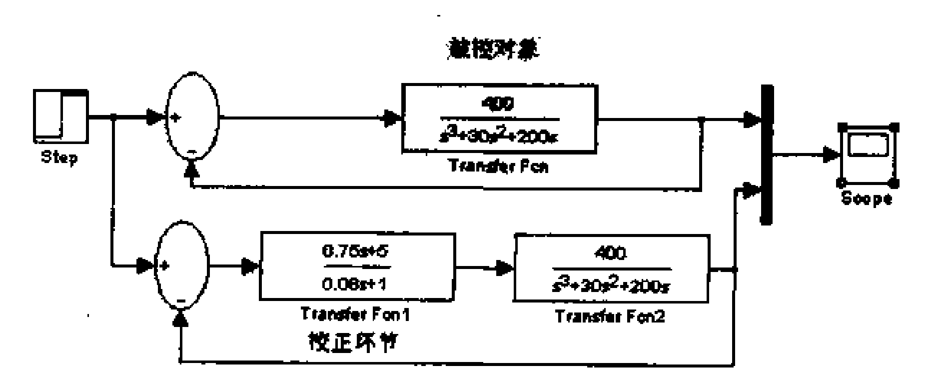
超前校正环节的传递函数为 $K(s)=K_c\dfrac{s+a}{s+b}$（$0<a<b$），其作用是使系统的相角裕度增加，从而提高系统的相对稳定性，扩大系统的频带。

在频域法中，采用伯特图进行设计是最常见的，基本思想是：改变原有系统开环频率特性，使其满足规定的低频增益和充分的稳定裕度。在伯特图设计方法中，为方便起见常常采用如下形式的传递函数的校正装置。

$$K(s)=K_c\frac{\alpha TS+1}{TS+1} \tag{12.2}$$

例 12.3 已知系统的传递函数为 $G(s)=\dfrac{400}{s(s^2+30s+200)}$**，系统的设计性能指标为：速度误差常数为 10，相角裕量为 45 度。**

控制器的设计不是本书的讨论范围，我们直接给出设计结果。最终的控制框图如图 12.7 所示。



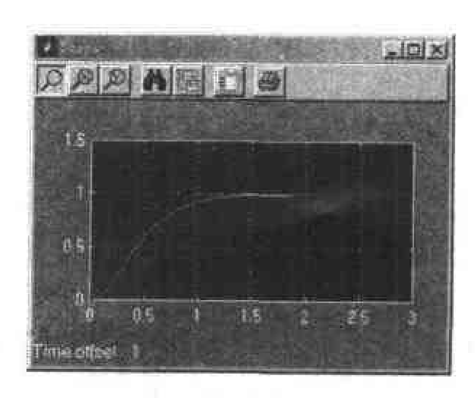


图 12.7　控制系统框图和仿真结果

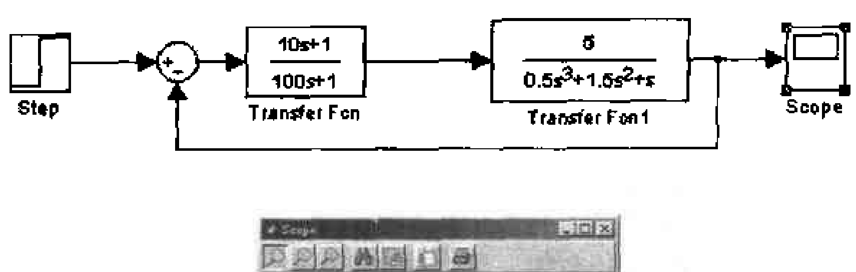
2. 滞后校正

如果校正装置中如果零点出现在极点之后，则相应的校正器为滞后校正。由于滞后校正装置给系统加入了滞后的相角，因而会使系统的动态稳定性降低。

滞后校正器可以降低系统的稳定性误差。同时使闭环系统的带宽降低，导致系统的动态响应速度变慢，这有利于减小外部噪声信号对系统的影响。

例 12.4：　已知系统的传递函数为 $G_0(s)=\dfrac{K}{s(s+1)(0.5s+1)}$**，要求控制系统满足以下性能指标：开环放大倍数** $K_v=5$**，相角裕度** $r(\omega_c)\geq 40^\circ$**，幅值裕度** $K_g\geq 10dB$。

控制系统的框图如图 12.8 所示。



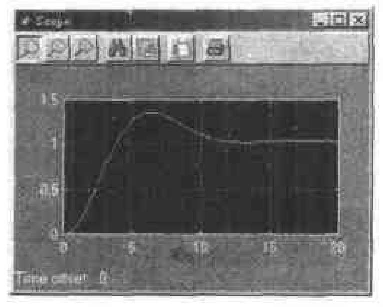


图 12.8　控制系统框图及仿真结果

12.3　现代控制系统的设计与仿真

12.3.1　现代控制系统的特点

现代控制理论以线性代数为数学工具，用状态空间方法描述系统内部的动力学性能，研究系统的稳定性、能控性、能观性等问题，用极点配置、最优控制、状态方块等理论研究设计控制系统。

现代控制理论是在经典控制理论的基础上发展起来的，主要有以下特点：

（1）现代控制理论是以多变量、线性及非线性系统为研究对象的。在近代的工业过程控制、飞行器控制等许多领域中，被控对象变得日益复杂，其中包括了多变量耦合问题、参数时变问题和非线性问题。现代控制理论正是为了处理这样的复杂控制系统而发展起来的。

（2）现代控制理论是以时域中的状态空间方法对系统进行数学描述，并在此基础上对系统进行各种定性和定量的分析以及希望的控制规律设计。

（3）现代控制理论以现代数学方法为主要分析手段，如线性代数、微分方程和微分几何等现代数学理论在最优控制、非线性系统的控制问题当中都有广泛的应用，甚至像模糊数学、混沌及神经网络等最新的数学方法也已经在许多控制领域得到应用。

（4）现代控制技术的研究是以计算机为主要计算和分析工具的，计算机技术的发展极大地促进了控制理论研究的广泛应用与推广。

（5）现代控制技术的研究对象还包括没有或不能用精确数学模型进行描述的广泛意义上的非线性系统，其中诸如神经网络控制、自适应控制等先进控制技术已成功应用到卫星、航天等大系统的控制当中，现代控制技术能够处理复杂非线性、相互耦合、外界干扰、多输入多输出、参数时变等复杂对象，研究的范围与经典控制理论相比极大地扩展了。

现代控制理论的内容十分丰富，而 MATLAB 软件中几乎包含了所有控制理论与控制系

统分析技术的内容，这为现代控制理论的研究和工程应用带来极大的方便。

本节将以现代控制理论中的模型参考自适应控制方法为例，通过实例来说明如何使用 MATLAB 及 SIMULINK 软件进行现代控制系统的分析与仿真。

12.3.2　模型参考自适应系统

在经典控制理论当中要设计一个性能良好的反馈系统，通常需要掌握被控系统的数学模型，然而，在实际工程中有一些被控对象的数学模型事先难以获得，或者模型的参数是时变的。对于这样的系统，一般的控制方法往往难以获得满意的性能。如果系统本身能够不断地测量被控对象的性能或参数，并反馈到控制器当中，就可以使系统运行在某种意义下的最优或次优状态。按照这样的思想建立的控制系统，我们称之为自适应控制系统。

自适应控制的思想首先在 20 世纪 50 年代提出，当时主要是针对具体的设计方案进行讨论，没有形成一个理论体系。60 年代现代控制理论的发展所取得的一些成果，例如状态空间方法、稳定性理论、最优控制、随机控制等，为自适应控制理论的形成和发展创造了条件。70 年代以来自适应理论的研究有了显著的进步，提出了一系列的控制方案，并出现了许多实际应用的例子。计算机性能的提高一方面为自适应控制的应用创造了技术条件，另一方面也促进了自适应控制理论的发展。

根据设计原理和结构的不同，自适应控制系统可分为变增益的自适应控制系统、模型参考自适应系统和自校正控制系统。

模型参考自适应控制系统由两个环路组成，内环由调节器和被控对象组成可调系统，外环由参考模型与自适应机构组成。参考模型代表某个优化指标。当被控对象受干扰作用影响使运行特性偏离了最优轨迹时，参考模型的输出与被控对象的输出相比较产生了一个广义误差 e。该误差通过自适应机构，根据一定的自适应规律产生反馈作用，以修正调节器的参数或产生一个辅助的控制信号，促使可调系统跟踪参考模型，使广义误差趋于极小。

模型参考自适应控制设计方法主要根据李亚普洛夫稳定性理论和超稳定理论，可以适用于线性和非线性系统，这样可以保证系统在稳定的前提下对系统的参数变化具有适应性，并提高有关的性能指标。

12.3.3　实例

下面我们研究的是美国国家航天局光学望远镜伺服系统的跟踪控制系统。该望远镜用于激光测距和光学通信实验，系统对动态跟踪精度要求很高，通常工作在三种方式：搜索方式、自动跟踪方式和程序跟踪方式。一般而言，在运行过程中，由于受到望远镜仰角的变化以及轴承的采用，负载惯量、扭矩和动静摩擦发生变化是不可避免的，因此采用常规的控制手段很难完成上述三个任务，于是将采用模型参考自适应控制方案。

望远镜的伺服控制系统如图 12.9 所示，其中 u 为自适应控制信号，f 代表主轴系统摩擦力的特性，它与机构的运行速度有明显的非线性关系，称之为库仑摩擦。

根据模型参考自适应理论，可以得到自适应控制信号 u 的形式为

$$u = K_1\theta_p + K_2 z + K_3\,\mathrm{sgn}\,\dot{\theta}_p$$

$$K_1(t) = B_1\int_0^t \dot{e}\dot{\theta}_p dt + C_1\dot{e}\dot{\theta}_p$$

$$K_2(t) = B_2\int_0^t \dot{e}z dt + C_2\dot{e}z$$

$$K_3(t) = B_3\int_0^t \dot{e}\,\mathrm{sgn}\,\dot{\theta}_p dt + C_3\dot{e}\,\mathrm{sgn}\,\dot{\theta}_p \tag{12.3}$$

（12.3）式中，k_1、k_2、k_3 为可调参数，它们由自适应机构调节；第三项用来补偿轴承摩擦的影响。

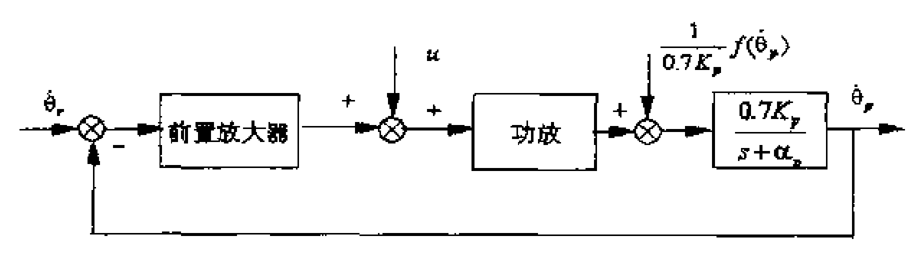


图 12.9　望远镜伺服系统的结构示意图

设参考位置输入为正弦信号，角速度为 0.2rad/s，幅值为 1.0rad/s，对象的传递函数为 6/（s+0.25）。通过仿真可以获得图 12.11 的位置跟踪曲线。同时，我们还可以实时改变模型中的参数，观察控制方法的鲁棒性。从实验结果可以发现，模型参考自适应控制方法对系统模型的依赖性要低于常规的控制方法，对于慢变化时变系统和某些非线性问题具有良好的鲁棒性；同时，它能够较好地克服低速情况下非线性摩擦（库仑）摩擦对控制系统带来的影响。

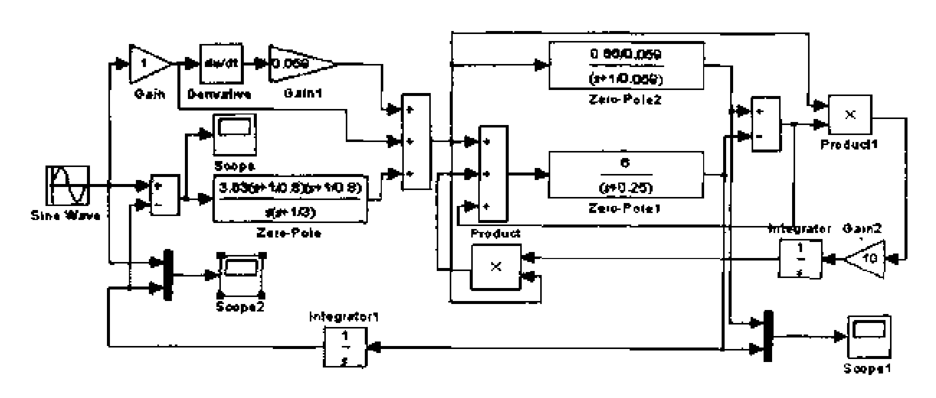


图 12.10　望远镜伺服系统的模型参考自适应跟踪控制方框图



图 12.11　正弦信号输入的位置跟踪曲线